MATCHING RESUMES WITH
JOB DESCRIPTIONS USING LATENT SEMANTIC INDEXING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF

ÇANKAYA UNIVERSITY

BY

MURAT POJON

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF

COMPUTER ENGINEERING

AUGUST 2014

Title of the Thesis: **Matching Resumes with Job Descriptions Using Latent Semantic Indexing**

Submitted by **Murat POJON**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

[illegible]

Prof. Dr. Taner ALTUNOK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Murat SARAN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Abdül Kadir GÖRÜR
Supervisor

Examination Date: 21.08.2014

Examining Committee Members

Assist. Prof. Dr. Abdül Kadir GÖRÜR	(Çankaya Univ.)	[illegible]
Assoc. Prof. Dr. Fahd JARAD	(THK Univ.)	[illegible]
Assist. Prof. Dr. Reza ZARE HASSANPOUR	(Çankaya Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	:	Murat POJON
Signature	:	[illegible]
Date	:	03.08.2014

ABSTRACT

MATCHING RESUMES WITH JOB DESCRIPTIONS USING LATENT SEMANTIC INDEXING

POJON, Murat

M.Sc., Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Abdül Kadir GÖRÜR

August 2014, 30 pages

In this thesis, Vector Space Model of Information Retrieval is examined. First, the classical method of term frequency inverse document frequency is presented as an introduction to the problem. After introducing basics, the thesis explains the concept of Latent Semantic Indexing. Singular Value Decomposition, which is the fundamental of Latent Semantic Indexing, is explained without going too deep into Linear Algebra. Relationship between Singular Value Decomposition and Latent Semantic Indexing is also explored. Finally, thesis presents the results of its demonstration, which is matching a Resume with an appropriate Job Description by using Latent Semantic Indexing and comparing it with the classical Vector Space method.

Keywords: Term Frequency, Inverse Document Frequency, Term Document Matrix, Latent Semantic Indexing, Singular Value Decomposition, Low Rank Approximation.

ÖZ

GİZLİ ANLAMSAL DİZİNLEME KULLANARAK ÖZ GEÇMİŞ VE İŞ İLANI EŞLEŞTİRMEK

POJON, Murat

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi: Yrd. Doç. Dr. Abdül Kadir GÖRÜR

Ağustos 2014, 30 sayfa

Bu tezde, Vektör Uzayı Modelli Bilgi Erişimi incelendi. İlk olarak, klasik Terim Frekansı Ters Doküman Frekansı metodu gösterildi. Temel kavramlar gösterildikten sonar, Gizli Anlamsal Dizinleme anlatıldı. Sonra, Gizli Anlamsal Dizinlemenin temeli olan Tekil Değer Ayrışımı, Lineer Cebire çok fazla girmeden anlatıldı. Tekil Değer Ayrışımı ve Gizli Anlamsal Dizinleme arasındaki ilişki gösterildi. Son olarak, tezin uygulamasının sonuçları sunuldu. Bu uygulamada, Gizli Anlamsal Dizinleme kullanılarak iş öz geçmişleri ve iş ilanları eşleştirildi ve klasik metodun sonuçları ile karşılaştırıldı.

Anahtar Kelimeler: Terim Frekansı, Ters Doküman Frekansı, Terim Doküman Matrisi, Gizli Anlamsal Dizinleme, Tekil Değer Ayrışımı, Düşük Kertesli Yaklaştırım.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Assist. Prof. Dr. Abdül Kadir GÖRÜR for his supervision, special guidance, suggestions, and encouragement through the development of this thesis.

TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM iii
ABSTRACT iv
ÖZ v
ACKNOWLEDGEMENTS vi
TABLE OF CONTENTS vii
LIST OF FIGURES viii
LIST OF TABLES ix
LIST OF ABBREVIATIONS x

CHAPTERS:

1. INTRODUCTION 1
   - 1.1. Information Retrieval 1
   - 1.2. Problem 2
   - 1.3. Approach 2
2. TERM VECTOR MODEL 4
   - 2.1. Definition 4
   - 2.2. Cosine Similarity 6
   - 2.3. Weighing Functions 7
3. LATENT SEMANTIC INDEXING 9
   - 3.1. Definition 9
   - 3.2. Singular Value Decomposition 9
   - 3.3. Low Rank Approximation with SVD 12
   - 3.4. Usage of SVD in Information Retrieval 14
   - 3.5. Geometric Justification of LSI 18
4. APPLICATION OF LSI TO THE PROBLEM 21
   - 4.1. Collecting the Documents 21
   - 4.2. Creating the Dictionary 22
   - 4.3. Creating the Document Vectors and TD Matrix 23
   - 4.4. Creating TFIDF and LSI Matrix 24
   - 4.5. Precision and Recall 26
   - 4.6. Testing the Results 28
   - 4.7. Software Tools Used 29
5. CONCLUSION 30

REFERENCES R1

APPENDICES A1

A. CURRICULUM VITAE A1

LIST OF FIGURES

FIGURES

Figure 1 Vectors on a 3D Coordinate System..19

Figure 2 Modifed Vectors on Coordinate System..20

Figure 3 Transfering Jobs into the Corpus..21

Figure 4 Tokenizing a Sentence..22

Figure 5 Stopwords are Removed...23

Figure 6 Stemming is Done on Words..23

Figure 7 Constructing a Document Vector...24

Figure 8 Overall Schema of the System...26

Figure 9 Set Notation of Retrieved and Relevant Documents................................27

LIST OF TABLES

TABLES

Table 1 Ranks on each Corpus Size..25

Table 2 Precision and Recall Values in the Corpus of Size 150............................28

Table 3 Precision and Recall Values in the Corpus of Size 300............................29

Table 4 Precision and Recall Values in the Corpus of Size 450............................29

LIST OF ABBREVIATIONS

IR	Information Retrieval
CV	Curriculum Vitae
TD	Term Document
TF	Term Frequency
IDF	Inverse Document Frequency
TFIDF	Term Frequency Inverse Document Frequency
LSI	Latent Semantic Indexing
SVD	Singular Value Decomposition
API	Application Programming Interface
PHP	Personal Home Page
ASP	Active Server Page

CHAPTER 1

INTRODUCTION

1.1 Information Retrieval

Information Retreval is defined as the activity of finding material(usually documents) of an unstructured nature(ususally text) that satisfies an information need from within large collections(usually stored on computers)[1]. Currently, it is mostly associated with search engines[2]. The user enters a text about the information needed, which is usually defined as “query”, and from its database of documents, the search engine returns a colection of documents that is deemed relevant to the query. Relevance is a value indicating how similar the document is to the query. Depending on the IR method used, similarity may be syntactic, meaning similarity based on the structure of words, or semantic, meaning similarity based on meaning of words, or a combination of both.

In order to process queries efficiently, a IR system needs to transform documents into suitable formats. There are different methods of Information Retrieval based on how they construct the documents and how they find the relation between query and the documents. This thesis will use a Linear Algebraic Model called Vector Space Model. Documents and queries will be represented as vectors, and a similarity function between the vectors will be used to determine the factor of relevance[3].

1.2 Problem

The problem addressed in this thesis is matching a Job Description with an appropriate Resume. We have a database contianing a relatively large number of Job Descirptions. When a CV is submitted, the program will find relevant Job Descriptions and show them to the user.

The problem of extracting information from a CV is well known, and numerous softwares have tried to address the issue. Notable ones include Sovren, Daxtra, TextKernel. Most of these and other similar softwares use the process known as CV parsing. This means converting a free-form CV/Resume into structured information suitable for storage, reporting and manipulation by a computer. CV Parsing is a smaller part of the discipline called Natural Language Processing, which is constucting a structured document from an unstrucutred text.

Observing these softwares and the algorithms show a common difficulty. In order to exract semantic information from a text, they need hand-written rules about the structure of text[4]. In the case of a CV Parser for example, they need to determine what type of information a body of text is referring to, does it refer to a school name, or a company or a residential address? To solve this problem, one needs a large database containing names of universities, city names and company names. This is known as an ontology. This can be a very tiresome task that needs constant updating. Furthermore, one needs a set of patterns that will reperesent key information on a CV. Again making such a patern list is a difficult process.

1.3 Approach

In this thesis, my main goal is to construct a method to match a CV and a Job description, using an algebraic information retrieval model. Here are the main steps of the project:

1.Collect a large body of Job Descriptions and Resumes. In the internet, a Job Description is much more easily found than a Resume. Therefore, Job Descriptions will be used as documents and Reumes will be used as queries.

2.Create a dictionary, a list of terms, from the Job Descriptions collected.

3.From each Job Description in the collection, construct a document vector representing the Job.

4.By using Latent Sematic Indexing, alter the values of the document vectors, so that vectors will represent the semantic structures of the documents.

5.Create a program that will take a CV/Resume as query and return Job Descriptions as results. Query CV will also be transformed into a document vector. And then the program will compute the similarity between the query vector and all the document vector. Most similar documents will be returned as results.

6.Calculate the overall efficiency of the system by determining whether the results are really similar to the query. This will be done according to the job categories of Resumes anf Job Descriptions. If the program bring job descriptions of similar category with the query CV, it will be deemed seccussfull.

7.Statistics about the results will be formed. It will observe how the efficiency of the program changes depending on the number of documents and the method used.

CHAPTER 2

TERM VECTOR MODEL

2.1 Definition

Term-Vector Model (or Vector Space Model) is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, in our case, index terms.

In Term-Vector model, each document is represented by an n-dimensional vector where n is the number of terms in our terms list[5]. What each dimension in the vector represents depends on the IR method used. For example , each dimensions can simply reperesent whether the that particaular term appears on the document or not. In that case value can eihter be 1 for true, or 0 for false. It can also represent how many times that term appears on document. In that case, the value can be any non-zero integer. More complicated methods of calculation can assign decimal or irrational values for dimensions, and some of these methods will be addressed later in this thesis.

Here is a smal example of term vector model:

Document 1 *: I have a blue car.*

Document 2 : *Cars have wheels.*

Document 3 : *Some cars are blue, some cars are red.*

Here is our small list of document. In order to transform these documents into vectors. We first need to create a dictionary of words. Let T be the list of terms(words).

T = {*I, have, a, blue, car, cars, wheels, some, are, red*}

Such a list will usually be considered inefficient, because it refers to singular and plural forms of "*car*" as different words, when they are actually the same word. To solve this issue, we need to perform stemming, which is the process of reducing a word to its root. Moreover, it includes very common words like "*I*", "*are*", "*some*". These words will not have much use in distinguishing documents. Words like these are called stopwords. There is no standard list or definition of stop words, but search engines usually remove the most common words if they are not doing a phrase search. Stemming and stopword remval are esential to constructing an efficient term-vector model. Without these, our term list may be too long and vectors can be too large, leading to slower retrieval speeds.

After applying stemming and stopword removal, we get our shortened list:

T = { *blue, car, wheel, red*}

Now that we have our list, we can now construct vectors from our documents. Three documents will be transformed into three 4-dimensional vectors where each dimension represents how many times the term occurs in that document.

Document 1 $\rightarrow V_1 = \{1, 1, 0, 0\}$

Document 2 $\rightarrow V_2 = \{0, 1, 1, 0\}$

Document 3 $\rightarrow V_3 = \{1, 2, 0, 1\}$

Now that our documents are transformed into vectors, we can do certain vectos operations to analyze the documents.

2.2 Cosine Similarity

Our main method of deciding the similarity of documents will be cosine similarity. It is the cosine of the angle between two vectors[6].

Here is how we derive the formula for calculating the value:

First, we need to calculate the dot product. Let V and W be two n-dimensional vectors. The dot product of two vectors $V \cdot W$ is the sum of $V_i \times W_i$ where $0 \leq i \leq n$ and V_i is the ith coordinate of the vector V_1. Formulated version is :

$$V \cdot W = \sum_{i=0}^{n} (V_i \times W_i) \tag{2.1}$$

Another value of a vector we need is length. Let V be a n-dimensional vector. The length of V, dentoed by |V| is formulated by :

$$\sqrt[2]{\sum_{i=0}^{n} V_i^2} \tag{2.2}$$

Since the dot product can also formulated by:

$$V \cdot W = |V||W| \cos\theta \tag{2.3}$$

where θ is the angle between V and W, final formula of the dot product can be written as :

$$\cos\theta = \frac{V \cdot W}{|V||W|} \tag{2.4}$$

In Term-Vector model, this formula indicates how similar two documents are. It can be a value between -1 and 1. For two documents D_1 and D_2 , if all the terms of D_1 and D_2 are common to both of them, then cosine similarity will be 1. If no two terms are common, then cosine similarity will be 0. Let D_1,D_2,D_3 be three document vectors where:

$$D_1 = \{1,0,2,0\}$$

$$D_2 = \{0,1,0,1\}$$

$$D_3 = \{2,0,4,0\}$$

Cosine similarities of documents will be :

$$Sim(D_1,D_2) = 0;$$

$$Sim(D_1,D_3) = 1;$$

$$Sim(D_2,D_3) = 0;$$

Notice the documents don't need to be same for cosine similarity to be 1.

2.3 Weighing Functions

So far, coordinate values of document vectors are calculated as how many times a term occurs in the document. This can be an inefficient way to construct the vectors. A term can occur a lot in a certain document and thus, have a high value in the document vector. However, if it occurs a lot in ever other document, then it will be wrong to assign significance to that word, since it will not give much information about the content of the document. Conversely, a terms that has a very low occurance in a document will not have much meaning in the calculations, but if it is a rare word in the document space, it will have a significance in determining the content of document.

To addres these issues, a weighing function must be used. In this thesis, we wil use the most common weighing function, which is "term frequency-inverse document frequency". In order to explain the formula, first we need to introduce certain concepts:

Let t be a term in our dictionary. If term t occurs in n different documents in our document space, then t has a document frequency of n.

Inverse document frequency idf is computed as[8]:

$$idf = \log \frac{N}{n+1} \tag{2.5}$$

where n is document frequency and N is the number of documents in outr list. Inverse document frequency is unique to every term.

Another thing we need to calculate is term frequency. It is essentially a modifed version of term occurence number. Let t be a term that occurs f times in document d. Then term frequency of t in d is[9]:

$$tf(t,d) = 0.5 + \frac{0.5 \times f}{m} \tag{2.6}$$

where f is the occurence number of t in d and m is the occurence number of the term with highest frequency in d. Term frequency is unique to every term-document couple.

Finally, we have what we need to calculate tf-idf[10]:

$$tfidf(t,d) = tf(t,d) \times idf(t) \tag{2.7}$$

We should look at the reasons for calculations in this formula. Main problem in unmodified calculation was that it does not take rarity of the term. Think of a document list where each document is a job description. The term job will occur in almost all documents. Therefore, the term job shouldn’t have much value when comparing documents. The less frequent a term is, more valuable it should be. Hence, the justification of inverse document frequency. And the reason for calculating term frequency is that we need to normalize term occurences for all documents. Longer documents will naturally have terms with more occurences. By dividing term occurence with the highest frequency, we make sure that differences in document size wil not cause too much variation in term frequency values.

CHAPTER 3

LATENT SEMANTIC INDEXING

3.1 Definition

Latent Semantic Indexing (or Latent Semantic Analysis) is a theory and method for extracting and representing the contextual usage meaning of words by statistical computations applied to document list[11]. The reason for the word Latent (hidden) is that the method doesn’t use any sematic process. It is a methematical process that enhances results semantically. We dont’t know what kind of sematic relation is constructed during the modification process, we can only find that by observing the results.

3.2 Singular Value Decompositon

Latent Sematic Indexing alters the term-document matrix with a linear algebraic method called Singular Value Decomposition[12].

Let A be m × n real matrix. Singular Value Decomposition states that there exist matrices U, S, V such that[13]:

$$A_{m\times n} = U_{m\times k} \times S_{k\times k} \times (V_{k\times n})^t \qquad (3.1)$$

where k is an integer between m and n, U and V are orthogonal matrices, meaning their transposes are equal to their inverses, and S is a diagonal matrix. Non-zero values of S are called the singular values of A.

There are different methods of computing the singular value decomposition of a matrix. We will observe one of them as an example.

First we need to introduce the conceopt of eigenvalue and eigenvector. Let A be a m×n matrix. Then there exists a vector V and a value λ such that[14]:

$$AV = \lambda V \tag{3.2}$$

In that case, V is called an eigenvector of A, and λ is called an eigenvalue of A[15]. Computation of eigenvalues and eigenvectors are beyond the scope of this thesis. Calculations will be done by computers and verified by the example.

Let A be a 2 × 2 matrix such that: $A = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix}$

Then by SVD tehorem, there must be 2×2 orthagonal matrices U, V and a 2×2 diagonal matrix S such that $A = U \times S \times V^t$

First, we must compute A × A^t and A^t × A. We find the matrices as:

$$A \times A^t = \begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix} \qquad A^t \times A = \begin{bmatrix} 2 & 5 \\ 5 & 13 \end{bmatrix}$$

Eigenvectors of A × A^t will form the colums of U, and eigenvectors of A^t × A will form the columns of V, and eigenvalues of either matrix will form squareroots of the diagonal entires of S[17].

Using the calculation done by computer, we find that eigenvectors of A × A^t are V_1 and V_2 such that V_1 = {0.5760,0.8174} and V_2 ={0.8174,-0.5760}. Eigenvectors of A^t × A are W_1 = {0.3606,0.9327} and W_2 = {0.9327,-0.3606}. Eigenvalues of any of matrices are λ_1= {14.9258} and λ_2= {0.067}.

Now we must verify the results for eigenvalues and eigenvectors.

For A × A^t :

$$(A \times A^t)V_1 = \begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix} \times [0.5760 \ \ 0.8174] = [8.6018 \ \ 12.2060]$$

$$\lambda_1 \times V_1 = 14.9258 \times [0.5760 \ \ 0.8174] = [8.5972 \ \ 12.2003]$$

$$(A \times A^t)V_1 \cong \lambda_1 \times V_1$$

$$(A \times A^t)V_2 = \begin{bmatrix} 5 & 7 \\ 7 & 10 \end{bmatrix} \times [0.8174 \quad -0.5760] = [0.055 \quad -0.0382]$$

$$\lambda_2 \times V_2 = 0.067 \times [0.8174 \quad -0.5760] = [0.0548 \ -0.0386]$$

$$(A \times A^t)V_2 \cong \lambda_2 \times V_2$$

For A^t × A :

$(A^t \times A)W_1 = \begin{bmatrix} 2 & 5 \\ 5 & 13 \end{bmatrix} \times [0.3606 \;\; 0.9327] = [5.3847 \;\; 13.9281]$

$\lambda_1 \times W_1 = 14.9258 \times [0.3606 \;\; 0.9327] = [5.3822 \;\; 13.9213]$

$$(A^t \times A)W_1 \cong \lambda_1 \times W_1$$

$(A^t \times A)W_2 = \begin{bmatrix} 2 & 5 \\ 5 & 13 \end{bmatrix} \times [0.9327 \; -0.3606] = [0.0624 \quad -0.0243]$

$\lambda_2 \times W_2 = 0.067 \times [0.9327 \; -0.3606] = [0.0625 \; -0.0242]$

$$(A^t \times A)W_2 \cong \lambda_2 \times W_2$$

Note that computations are done with 4 point decimals, so some precision will be lost. But the values are close enough to confirm the eigenvlaues and eigenvectors.

Using these vectors, we get our matrices U, S and V such that:

$$U = \begin{bmatrix} 0.5760 & 0.8174 \\ 0.8174 & -0.5760 \end{bmatrix}$$

$$S = \begin{bmatrix} 3.8643 & 0 \\ 0 & 0.2588 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.3606 & 0.9327 \\ 0.9327 & -0.3606 \end{bmatrix}$$

To confirm these matirces, we compute A using the SVD equation:

$U \times S \times V^t = \begin{bmatrix} 0.5760 & 0.8174 \\ 0.8174 & -0.5760 \end{bmatrix} \times \begin{bmatrix} 3.8643 & 0 \\ 0 & 0.2588 \end{bmatrix} \times \begin{bmatrix} 0.3606 & 0.9327 \\ 0.9327 & -0.3606 \end{bmatrix}$

$U \times S \times V^t = \begin{bmatrix} 0.9999 & 1.9998 \\ 1.0001 & 2.9998 \end{bmatrix}$

$U \times S \times V^t \cong A$

Again with 4 decimal points, the values are close enough to confirm SVD equation.

3.3 Low Rank Approximation with SVD

Our main reason to use singular value decomposition is low rank approximation[18]. In order to show what it is, we need to introduce some new concepts.

Let A be a 2×2 matrix such that : $A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$

If we create vectors out of the columns of A, we get V_1 = {1,2}, V_2 = {3,1}. These are called the column vectors of A. The vector space that is formed by these two vectors are called the column space of A. And dimension of that space is called the column rank of A[19].

Vectors V_1 and V_2 are linearly independent, meaning that one can't be written as a multiple of other. This means that any 2-dimensional vector can be written as a linear combination of V_1 and V_2. So these vectors from a vector space of two dimensions. This means A has a column rank of 2. Using the same logic with rows of A, we can find that A has a row rank of 2. Minimum of the row rank and column rank of A will be the rank of A, which is 2.

Let us consider another matrix B such that : $B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$

Column vectors of B are V_1 = {1,2} and V_2 = {2,4}. Since V_1 and V_2 are multiples of each other, any linear combination of these two vectors will be just an extension of the vector {1,2}. So the vector space formed by V_1 and V_2 is a line, meaning the rank of B is 1, despite the fact that B is a 2×2 matrix.

Now that we know what rank is, we can show how singular value decomposition is used for low rank approximation.

Let A be a m×n matrix with rank r, and U, S,and V be its singular value decomposition matrices. Let k be a number smaller than r. Using k, we will modify the matrices U, S and V. U_k will be the matrix formed by taking the first k columns of U. S_k will be a k×k diagonal matrix formed by the first k diagonal values of S. And V_k will be the matrix formed by the first k columns of V. With these matrices, we will calculate the modified version A_k of A such that :

$$A_k = U_k \times S_k \times (V_k)^t \tag{3.3}$$

The matrix A_k is an approximation of A with rank k. It still has the same size with A, but it has a lower dimension[20].

To better understand the concept of low rank approximation, we shall have an example.

Let X be a matrix such that:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0.2 \end{bmatrix}$$

Using singular value decomposition, we obtain matrices U, S and V.

$$U = \begin{bmatrix} 0.40 & -0.71 & -0.58 \\ 0.40 & 0.71 & -0.58 \\ 0.82 & 0 & 0.57 \end{bmatrix}$$

$$S = \begin{bmatrix} 1.74 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.11 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.70 & -0.71 & -0.07 \\ 0.70 & 0.71 & -0.07 \\ 0.09 & 0 & 1 \end{bmatrix}$$

We want to obtain a rank 2 approximation of the matrix X. So with k =2, we will find the matrices U_k,S_k,V_k such that:

$$U_k = \begin{bmatrix} 0.40 & -0.71 \\ 0.40 & 0.71 \\ 0.82 & 0 \end{bmatrix}$$

$$S_k = \begin{bmatrix} 1.74 & 0 \\ 0 & 1 \end{bmatrix}$$

$$V_k = \begin{bmatrix} 0.70 & -0.71 \\ 0.70 & 0.71 \\ 0.09 & 0 \end{bmatrix}$$

Using the modified matrices, we compute X_k, which is:

$$X_k = U_k \times S_k \times (V_k)^t$$

$$X_k = \begin{bmatrix} 0.40 & -0.71 \\ 0.40 & 0.71 \\ 0.82 & 0 \end{bmatrix} \times \begin{bmatrix} 1.74 & 0 \\ 0 & 1 \end{bmatrix} \times \left(\begin{bmatrix} 0.70 & -0.71 \\ 0.70 & 0.71 \\ 0.09 & 0 \end{bmatrix} \right)^t$$

$$X_k = \begin{bmatrix} 1 & 0 & 0.07 \\ 0 & 1 & 0.07 \\ 1 & 1 & 0.13 \end{bmatrix} \cong X$$

X_k is approximate to X, but it has a rank of 2.

3.4 Usage of SVD in Information Retrieval

Now that we know what singular value decomposition is, and we know how it is used in low rank approximation, we will look to its usage in Latent Sematic Indexing.

We will have an example with a small document corpus:

d1:*Php is a serverside web programming language.*

d2:*Web applications can be programmed with ASP.NET.*

d3:*ASP.NET applications can be written in C# or VB.NET languages.*

d4:*Unlike php, javascript is a clientside language.*

Here are 4 short documents about web programming. Normally, a corpus of 4 documents probably will not produce a semantically enhanced result with singular value decomposition. But this example is specially constructed so that singular value decomposition will give meaningfull results.

With stemming and stopword removal, we obtain our shortened term list T.

T = { *"php", "serverside", "web", "program", "language", "applications", "asp.net", "written" , "c#" , "vb.net" , "javascript" , "clientside"*}

Then we get our term-document matrix A, which is a 4×12 matrix.

$$A = \begin{bmatrix} 1{,}00 & 0{,}00 & 0{,}00 & 1{,}00 \\ 1{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 1{,}00 & 1{,}00 & 0{,}00 & 0{,}00 \\ 1{,}00 & 1{,}00 & 0{,}00 & 0{,}00 \\ 1{,}00 & 0{,}00 & 1{,}00 & 1{,}00 \\ 0{,}00 & 1{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 1{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 1{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 1{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 1{,}00 \end{bmatrix}$$

Using singular value decomposition, we obtain matrices U, S and V:

$$U = \begin{bmatrix} 0{,}30 & -0{,}47 & -0{,}17 & -0{,}03 \\ 0{,}18 & -0{,}25 & 0{,}15 & -0{,}47 \\ 0{,}34 & -0{,}17 & 0{,}48 & 0{,}05 \\ 0{,}34 & -0{,}17 & 0{,}48 & 0{,}05 \\ 0{,}50 & -0{,}18 & -0{,}37 & -0{,}25 \\ 0{,}36 & 0{,}37 & 0{,}12 & 0{,}30 \\ 0{,}36 & 0{,}37 & 0{,}12 & 0{,}30 \\ 0{,}20 & 0{,}29 & -0{,}21 & -0{,}22 \\ 0{,}20 & 0{,}29 & -0{,}21 & -0{,}22 \\ 0{,}20 & 0{,}29 & -0{,}21 & -0{,}22 \\ 0{,}11 & -0{,}22 & -0{,}32 & 0{,}44 \\ 0{,}11 & -0{,}22 & -0{,}32 & 0{,}44 \end{bmatrix}$$

$$S = \begin{bmatrix} 3{,}00 & 0{,}00 & 0{,}00 & 0{,}00 \\ 0{,}00 & 2{,}22 & 0{,}00 & 0{,}00 \\ 0{,}00 & 0{,}00 & 1{,}92 & 0{,}00 \\ 0{,}00 & 0{,}00 & 0{,}00 & 1{,}17 \end{bmatrix}$$

$$V = \begin{bmatrix} 0{,}55 & -0{,}56 & 0{,}30 & -0{,}55 \\ 0{,}46 & 0{,}19 & 0{,}62 & 0{,}61 \\ 0{,}60 & 0{,}64 & -0{,}39 & -0{,}26 \\ 0{,}34 & -0{,}49 & -0{,}61 & 0{,}51 \end{bmatrix}$$

After obtaining U,S and V, we want to compute a lower rank approximation of A. In this case, our rank will be 2.

We will take first two columns of U and V, and first two diagonal values of S to obtain U_2 , S_2 , V_2 such that:

$$U_2 = \begin{bmatrix} 0,30 & -0,47 \\ 0,18 & -0,25 \\ 0,34 & -0,17 \\ 0,34 & -0,17 \\ 0,50 & -0,18 \\ 0,36 & 0,37 \\ 0,36 & 0,37 \\ 0,20 & 0,29 \\ 0,20 & 0,29 \\ 0,20 & 0,29 \\ 0,11 & -0,22 \\ 0,11 & -0,22 \end{bmatrix} \qquad S = \begin{bmatrix} 3.00 & 0 \\ 0 & 2.22 \end{bmatrix} \qquad V_2 = \begin{bmatrix} 0,55 & -0,56 \\ 0,46 & 0,19 \\ 0,60 & 0,64 \\ 0,34 & -0,49 \end{bmatrix}$$

Then we compute our LSI modifed term-document matrix by the formula of

$$A_2 = U_2 \times S_2 \times (V_2)^t$$

(3.4)

and we get our result as :

$$A_2 = \begin{bmatrix} 1,08 & 0,22 & -0,13 & 0,82 \\ 0,61 & 0,15 & -0,02 & 0,46 \\ 0,76 & 0,40 & 0,38 & 0,53 \\ 0,76 & 0,40 & 0,38 & 0,53 \\ 1,05 & 0,62 & 0,64 & 0,71 \\ 0,13 & 0,65 & 1,18 & -0,04 \\ 0,13 & 0,65 & 1,18 & -0,04 \\ -0,02 & 0,40 & 0,78 & -0,11 \\ -0,02 & 0,40 & 0,78 & -0,11 \\ -0,02 & 0,40 & 0,78 & -0,11 \\ 0,46 & 0,07 & -0,11 & 0,36 \\ 0,46 & 0,07 & -0,11 & 0,36 \end{bmatrix}$$

Now we will submit a query to test the differences between the original matrix and LSI modifed matrix. Our query text will be “web programming”. First, we will construct our query vector just as we construct any document vector.

$$Q^t = [0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

Then we will look at the cosine similarities between query vector and each document vector to get the relevant results.

In original term-document matrix, only d1 and d2 will be returned as a relevant result because only these documents have common terms with the query. Any other document will have a cosine similarity of 0.

d1:*Php is a serverside web programming language.*

d2:*Web applications can be programmed with ASP.NET.*

In LSI modified matrix, however, we have more documents with non-zero cosine similarity. Here are the cosine similarities of 9 documents:

d1=0,52 d2=0,39 d3=0,23 d4=0,50

It brings all five documents with relevant results:

d1:*Php is a serverside web programming language.*

d2:*Web applications can be programmed with ASP.NET.*

d3:*ASP.NET applications can be written in C# or VB.NET languages.*

d4:*Unlike php, javascript is a clientside language.*

In the original term-document matrix, value of term “web” and “program” in the third and fourth documents was zero, because term didn’t occur in these docments. However, in the LSI modified matrix, value of the term has been increased. This is because d3 and d4 has common words with documents d1 and d2 and these documents contain the word “web” and “program”. This is the main principle of Latent Semantic Indexing. Its algorithm is based on the assumption that documents with a lot of common words will be close to each other in meaning, and document will have a sematic connection to an absent term if it is related to a document containing that term[21]. Converse is also true. If two documents have very few common words, then they will be semantically unrelated, and the terms in one document will have less semantic relation to the other document. This is the reason why some of the entries in the LSI modified matrix has negative values. The orignal value was zero, meaning that word didn’t occur in that document, but in the LSI matrix, value was less than zero, meaning that the word is even further away to document in terms of meaning.

With just a few documents, the assumption that common words imply similar meaning may give undesirable results. After all, just because there are common words between two documents doesn’t mean that other words will be semanitcally closer. But LSI will be practiced with a large amount of documents. A term value in a document will be modified according to every other document, and each of these documents will have a positive or negative effect on term value. The cumulative result of all these modification will be much meaningful semantically.

Latent Semantic Indexing solves two main problems that occur in classic term-vector model. One is synonymy and the other is polysemy[22]. Synonymy is when two different words have the same meaning, and polysemy is when a single word can have multiple meanings[23]. However, the word Latent, which means hidden, should remind that LSI modification doesn’t identify any polysemy or synonymy, it just enhances the corpus semantically so that problems caused by polysemy and synonymy will be reduced. LSI will never specifically tell us why a term not occuring on a document has a value on LSI modified matrix. LSI operation is mathematical, its results are semantic.

3.5 Geometric Justification of LSI

So far, we used singular value decompositon to sematically enhance the term document matrix, but we didn’t answer an important question. Why are we using singular value decomposition? What is the realtion between low rank approximation and semantics of terms and documents? To answer these questions, we will have a small exapmle.

Let A be a 3×4 matrix such that:

$$A = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 0 & 0.1 & -0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the columns of A, we can form 4 vectors with dimension of 3. Vectors are

V_1 = {2,0,0}, V_2 = {2,0.1,0}, V_3 = {1,-0.1,0}, V_4 = {0,0,1}

In order to understand the geometric effect of SVD better, we must show these vectors on a 3-dimensional coordinate system.

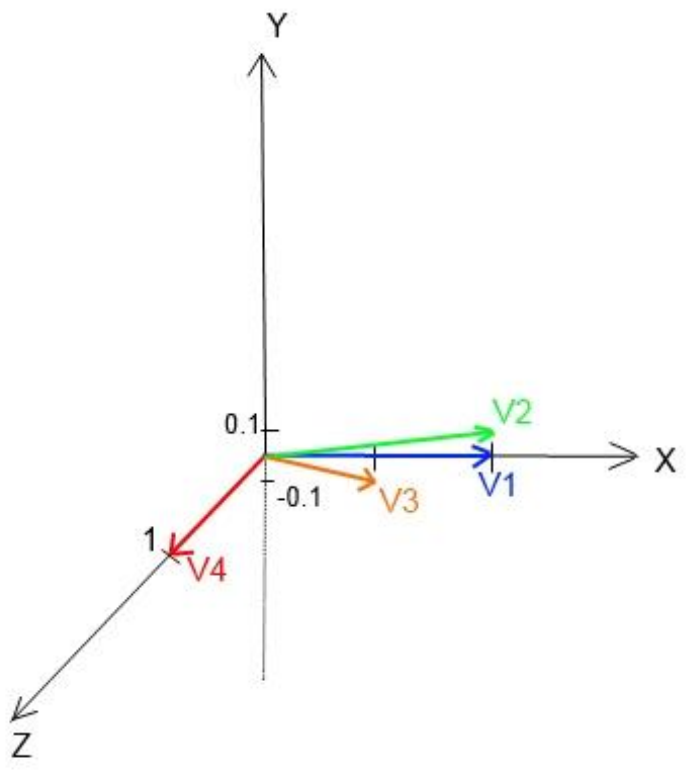

Figure 1: Vectors on a 3D Coordinate System

As we can see from the figure, V_1, V_2 and V_3 are in the same plane, whick is xy-plane, and V_4 is perpendicular to that plane. This means the 4 vectors form a 3-dimensional vector space, therefore the rank of matrix A is 3.

If we use singular value decomposition find a rank 2 approximation of A, we will get the matrix A_2.

$$A_2 = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 0.02 & 0.02 & 0.01 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The new vectors constructed from the rows of A_2 are W_1 = {2,0.02,0}, W_2 = {2,0.02,0}, W_3 = {1,0.01,0}, W_4 = {0,0,1}

Lets draw the new vectors on the coordinate system.

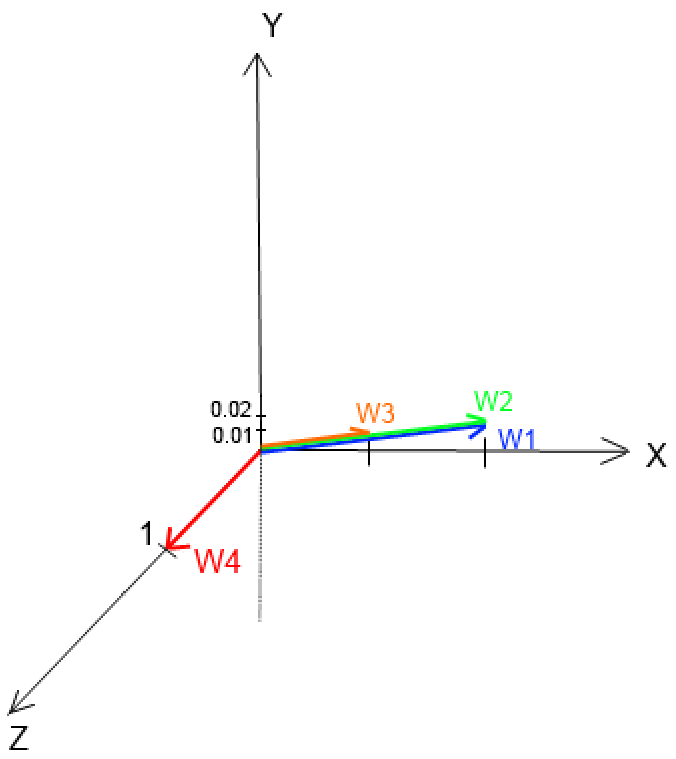

Figure 2: Modifed Vectors on Coordinate System

Now, the vectors W_1, W_2 and W_3 are on the same line and the four vectors form a plane, therefore rank of A_2 is 2. V_1, V_2 and V_3 had common non-zero coordinates and were close to each other in terms of angle. Vector V_4 on the other hand, had no common non-zero coordinates with the other vectors and was apart from the other vectors. So , if we want to have four vectors that are similar to the original one to form a 2-dimensional plane, logical choice would be to move vectors V_1, V_2 and V_3 together instead of moving V_4 to the three vectors. And this is what SVD does, to find a lower rank matrix that is as similar as possible to the original matrix. This translates into the main principle of Latent Semantic Indexing. If we think of vectors V_1, V_2 ,V_3, V_4 as documents, then the common non-zero coordinates will be common terms. Vectors with common non-zero coordinates move closer, just as documents with common terms become more similar.

CHAPTER 4

APPLICATION OF LSI TO THE PROBLEM

With the fundamental concepts about Latent Sematic Indexing is introduced, we can now use LSI for the task of matching Resumes with Job Descriptions.

4.1 Collecting the Documents

First step of the project was to collect Resumes and Job Descriptions. Resumes were hard to find in large numbers open in the web, so various sources are used to collect them. Job descriptions are relatively easier to find, so they are used to create terms and documents instead of CVs. All the job descriptions were obtianed from www.monster.com. A random sample of 450 Job Descriptions are obtained and put into database. In monster.com, Job Descriptions are divided into categories for more efficient search. These categories define the ganeral area of the job. For each Job Description, its text content and category is obtained and stored in the database.

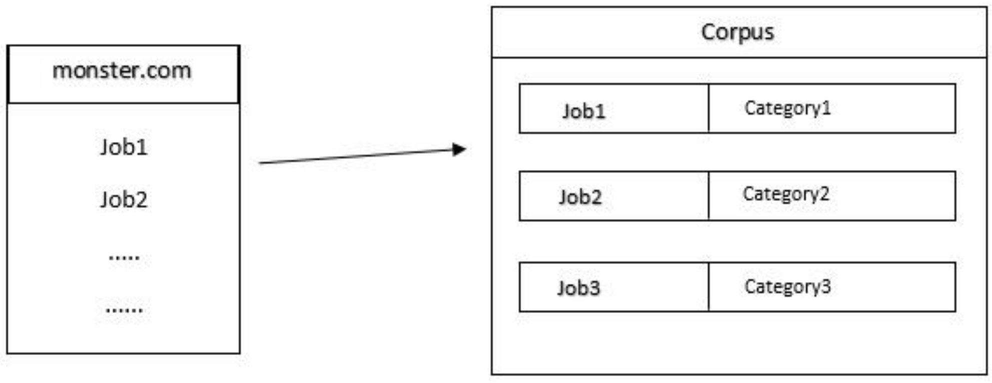

Figure 3: Transfering Jobs into the Corpus.

This operation is done manually, by copying the text content from the webpages and inseting it to a database table. At the end of this operation, a list of 450 job descriptions have been collected, each element in the list consisting of a text describing the job, and a label describing its category. 450 jobs in the list contians 12 different categories which are Administration, Architect, Banking, Creative Design, Economics, Editorial, Education, Engineering, Human Resources, Information Technology, Legal and Marketing.

In order to test the success of the system with different corpus sizes, three seperate term & document lists are created, first one created from 150 job descriptions, second one from 300, and the third one is created from all 450 job descriptions collected. With each list, a series of operation are made:

4.2 Creating the Dictionary

Dictionary is the list of words contained in the corpus. In order to create an efficient dictionary, three main operations are made: tokenization, stop word removal and stemming. Tokenization is the act of seperating a text into smaller parts, called tokens. In this case, tokens are words. Here is an example of tokenizing a sentence:

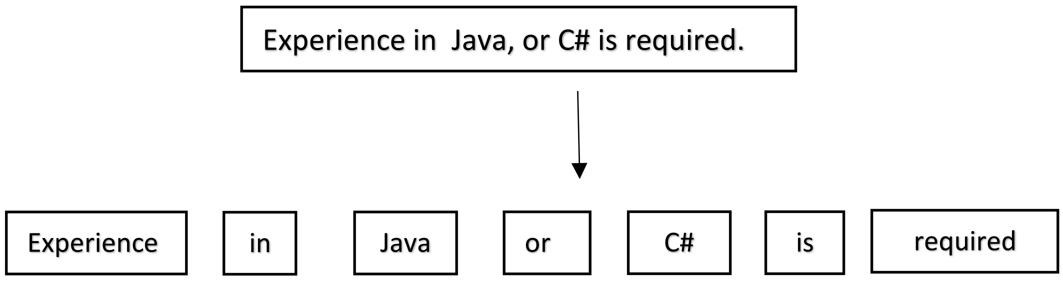

Figure 4: Tokenizing a Sentence.

After splitting the document into tokens, stop words should be removed. Stopwords are words that are very common in every text and doesn’t have any importance in determining the meaning of the text. Words like and,or, in, is are included in almost every stopword list, but in some lists, words like have, get, very are also included.

Experience
Java
C#
required

Figure 5: Stopwords are Removed.

After removing the stop words, the final operation is the stemmimg, which is reducing a word to its roots. In this project, Porter Stemming Algorithm is used. An additional operation during stemming is converting words into lower key.

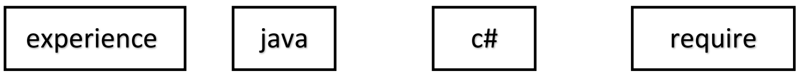

Figure 6: Stemming is Done on Words.

After these operations, words are added into the dictionary. An additional information that is added was the document frequency of the terms, meaning the number how many documents the word occured in. Terms that occur only once, terms that occur only in one document, and terms that occur in every document is removed from dictionary, because these words will not have any function in distinguisihing between the documents. It is possible that some of the removed terms might be useful, but shortening the list was necessary to build a system that will have a reasonable performance, even if it is just for demonstration.

4.3 Creating the Document Vectors and TD Matrix

Second task was to construct the document vectors. If the dictionary has n terms, then each job description was transformed into an n-dimensional vector. Here is an example of converting a small document into a vector:

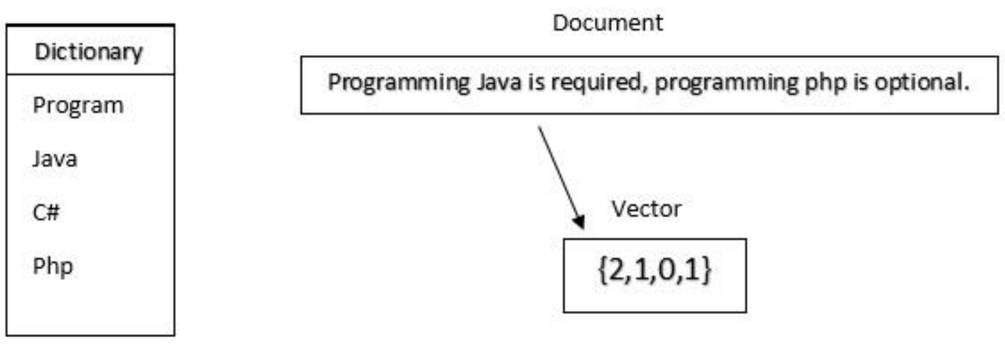

Figure 7: Constructing a Document Vector.

If there is n terms and m documents, this operation will result in creating m vectors which are all n-dimensional. By combining these vectors, an m x n matrix is constructed. We will call this matrix as TD (term-document) matrix.

4.4 Creating TFIDF and LSI Matrix

TD matrix is the first one created. In the second matrix, a weighing function of tf-idf (term frequency-inverde document frequency) will be used to obtain more efficient results.

Here is the original formulas used in previous examples:

$$idf \ = \log \frac{N}{n+1} \tag{4.1}$$

$$tf(t,d) = \ 0.5 + \frac{0.5 \times f}{m} \tag{4.2}$$

$$tfidf(t,d) = tf(t,d) \ \times idf(t) \tag{4.3}$$

Slight modifications are necessary when to use them in the project. In the original formula for idf, we have n+1 as denominator where n is the document frequency of the term. Reason for +1 was to avoid dividing by zero. But it is impossible to have n as zero. n=0 would mean that the term occurs in none of the documents, but such a term wouldn’t be in the dictionary in the first place. So the idf equation is modified as:

$$idf = \log\frac{N}{n} \tag{4.4}$$

The other modification will be made to tf formula. In the original formula, 0.5 were added to calculate term frequency. Such an addition prevents any zero values in the matrix and makes the calculations more difficult. Moreover, it causes values to be too similar, making it difficult to distinguish documents. Therefore, the addition of 0.5 is removed and modified formula is obtained:

$$tf(t,d) = \frac{0.5 \times f}{m} \tag{4.5}$$

Having the new formulas, weighted values are calcultated and the second matrix is obtained, which is called TFIDF (term frequency inverse document frequency) matrix.

Next part is to use Latent Semantic Indexing to modify the TFIDF matrix. Main aim of the project is to compare the results of TFIDF method and LSI method. So the LSI matrix will be created by modifying the TFIDF matrix, not TD matrix.

Details of Latent Semantic Indexing is explanied in chapter 3, and will not be explained here again. It is sufficient to say that LSI will be used to compute reduced rank versions of TFIDF matrix. An m x n TFIDF matrix with rank r will be converted to an m x n LSI matrix of rank k where k < r. In this stage, 3 matrices will be calculated each with different ranks. Reason for this is find which is the optimal rank for LSI to be useful. Here are the ranks for each diffferent size of corpus.

Corpus Size	Rank1	Rank2	Rank3
150	37	74	111
300	75	150	225
450	112	224	336

Table 1: Ranks on each Corpus Size

4.5 Preicison and Recall

Now, all the matrices have been obtained. Next step is to submit the queries and test the results. Each CV/Resume will be submitted as a query text. From the CV text, the query vector will be constructed. Depending on the method to use (TD,TFIDF or LSI) the query vector will be further modified. If the method is TFIDF or LSI, the query will be modified with tf-idf function. If the method is TD, then no modification will be made. After modified query is obtained, the cosine similarity between query vector and each of the document vectors will be calculated. All the documents will sorted by similarity value, and the top ten documents will be brought as result. Below is an example demonstrating how the system works:

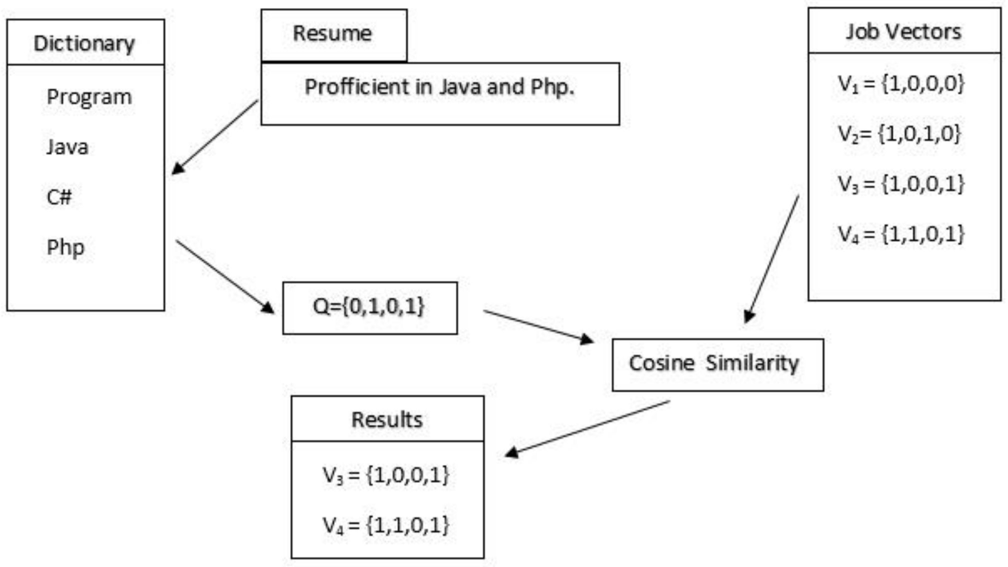

Figure 8: Overall Schema of the System.

Now, we can test the results. Verifying whether we have matching results or not is done by comparing the Categories of CV and Job descritpions. Each job description has only one category while a CV can have more than one category, snice a CV can be used to apply different types of jobs, while a job description is amde for one specific job. To measure the success of the query, we need to introduce two concepts, which are precision and recall.

Lets say a search engine or a similar Information Retrieval algorithm responds to a query, and brings results from a document collection.

We can divide the documents as relevant or irrelevant. Relevant documents are the ones that are supposed to be returned, and irrelevant documents are the ones that sholudn't be returned. We can also divide the documents as the ones that are retrieved, and the ones that are not retrieved. Below is an illustration of these classes with a set notation:

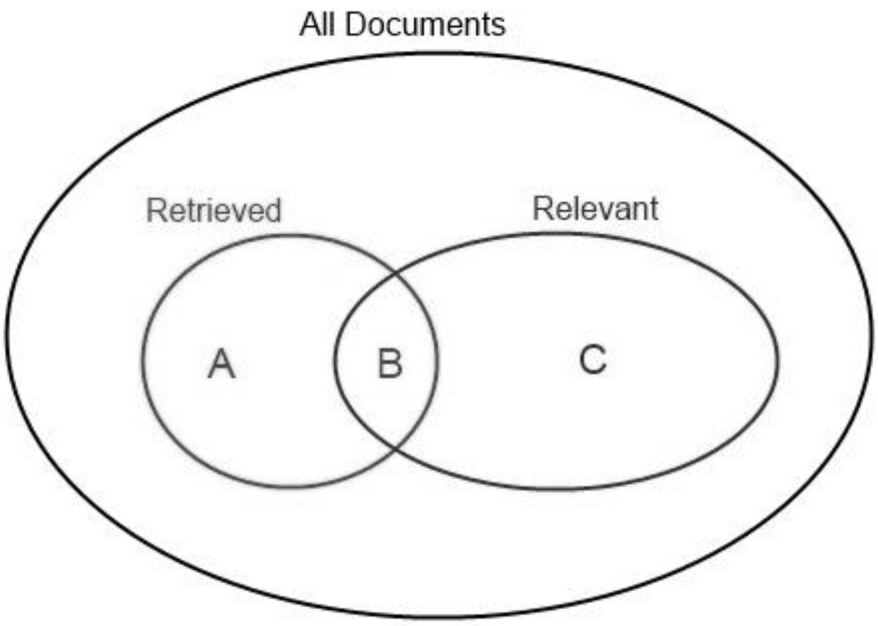

Figure 9: Set Notation of Retrieved and Relevant Documents

Using these sets, we can formulate precision and recall. Let n(X) denote the number of elements of set X. Then precision and recall are:

$$Precision = \frac{n(B)}{n(A \cup B)} \tag{4.6}$$

$$Recall = \frac{n(B)}{n(C \cup B)} \tag{4.7}$$

So, precision is the percentage of relevant results among the retrieved documents, and recall is the percentage of retrieved result among the relevant. In our case, relevant document means document with same catagory as the query.

Lets have an example. Let our query CV has the category of Economics. Suppose our query returns five results with 3 category of Economics, 1 Education and 1 Information Technology. Also, suppose that our document collection contains 10 documents with category of Economics. In that case:

Number of retrieved documents are 5.

Number of relevant documents are 10.

Number of relevant and retrieved documents are 3.

So precision and recall are:

$$P = \frac{3}{5} = 60\%$$

$$R = \frac{3}{10} = 30\%$$

4.6 Testing the Results

In order to compare the efficiencies of different methods, we need to compute the average preicison and recall values for all queries. Our three methods are TD, original unmodified term-document matrix, TFIDF, weighted matrix and LSI, SVD modified matrix. LSI matrix will have three different versions with three different ranks.

Here are the average precision and recall values for different methods. We have three result sets for different corpus sizes of 150, 300 and 450. In each cell, value before the slash is precision, value after slash is recall.

For size 150:

TD	TFIDF	LSI-37	LSI-74	LSI-111
0,58/0,29	0,61/0,34	0,67/0,37	0,59/0,31	0,61/0,33

Table 2: Precision and Recall Values in the Corpus of Size 150

For size 300:

TD	TFIDF	LSI-75	LSI-150	LSI-225
0,59/0,19	0,71/0,24	0,76/0,25	0,70/0,23	0,72/0,23

Table 3: Precision and Recall Values in the Corpus of Size 300

For size 450:

TD	TFIDF	LSI-112	LSI-224	LSI-336
0,66/0,14	0,73/0,17	0,81/0,17	0,78/0,16	0,76/0,16

Table 4: Precision and Recall Values in the Corpus of Size 450

Looking at these results, we draw certain conclusions:

1.Unmodified term-document matrix is the least efficient method out of three. In all corpus sizes, it has the worst performance.

2.Term frequency-inverse document frequency method gives better results than TD matrix, but in each corpus size, LSI with an appropriate rank provides better reults.

3.In all three corpus size, LSI with the lowest rank provides the best result, this means that the optimum rank for LSI is low compared to the original rank.

4.As the corpus size increases, LSI improves better than TFIDF. In the corpus size 450, all three LSI merhods have better results than TFIDF.

4.7 Software Tools Used

Here are the softwares that is used. The project is an ASP.NET web application written in C# programming language. In order to store the terms, documents and matrices, an SQL Server database is used. Matrix operations are done by an API called DotNetMatrix.

CHAPTER 5

CONCLUSION

Aim of this project was to address the issue of Resume-Job matching, and to find a way that can overcome semantic obstalces like synonymy and polysemy, and also a way that is easy to build. Latent Semantic Indexing is an appropriate way to solve this problem. Its latent nature allows it to make a semantic enhancement without the difficult operations of Natural Language Processing. The process was relatively easy to build, all that is needed was the Resumes and Job Descriptions to construct the documents. Aim of this thesis was to introduce and explain the Concept of Latent Semantic Indexing, explain the mathematical basis for it, which is Singular Value Decomposition, show relation between the literary basis of LSI and the linear algebraic basis of SVD. Furthermore as a demonstration, it is shown that LSI indeed does provide an improvement over the traditional method of term frequency inverse document frequency. Statistics provided show that LSI gives better results than TFIDF, and it is not much more difficult to construct. In the end, thesis demonstrated an efficient way to match diiferent bodies of text without ingnoring semantic relaitons.

REFERENCES

1. **Christopher D. M., Prabhakar R., Hinrich S., (2009),** "*An Introduction to Information Retrieval", Cambridge University Press, Cambridge, pp.38.

2. **Christopher D. M., Prabhakar R., Hinrich S., (2009)** "*An Introduction to Information Retrieval*", Cambridge University Press, Cambridge, pp.38.

3. www.inf.ed.ac.uk/teaching/courses/inf1/da/2013-2014/slides/inf1-da-13-16.pdf, (Data Download Date : 18-08-2014).

4. **Prakash M. N., Lucila O. M., Wendy W. C., (2011),** "*Natural language processing: an introduction*", J. Am. Med. Inform. Assoc., vol. 18, pp.1.

5. www.inf.ed.ac.uk/teaching/courses/inf1/da/2013-2014/slides/inf1-da-13-16.pdf, (Data Download Date : 18-08-2014).

6. www.inf.ed.ac.uk/teaching/courses/inf1/da/2013-2014/slides/inf1-da-13-16.pdf, (Data Download Date : 18-08-2014).

7. **Gerard S., Christopher B., (1988),** "*Term Weighting Approaches in Automatic Text Retrieval*", Information Processing & Management, vol. 24, pp.513.

8. **Deepak I. M., Shylaja K. R., Ravinandan M. E., (2014),** "*Development of Secure Multikeyword Retrieval Methodology for Encrypted Cloud Data*", International Journal of Science and Research, vol. 3, pp.496.

9. **Deepak I. M., Shylaja K. R., Ravinandan M. E., (2014),** "*Development of Secure Multikeyword Retrieval Methodology for Encrypted Cloud Data*", International Journal of Science and Research, vol. 3, pp.496.

10. **Deepak I. M., Shylaja K. R., Ravinandan M. E., (2014),** "*Development of Secure Multikeyword Retrieval Methodology for Encrypted Cloud Data*", International Journal of Science and Research, vol. 3, pp.496.

.

11. **Thomas K. L., Peter W. F., Darrell L., (1998),** "*An Introduction to Latent Semantic Analysis", *Discourse Processes, vol. 25, pp.2.

12. **Thomas K. L., Peter W. F., Darrell L., (1998),** "*An Introduction to Latent Semantic Analysis", *Discourse Processes, vol. 25, pp.2.

13. www.ling.ohio-state.edu/~kbaker/pubs/ Singular_Value_Decomposition_ Tutorial.pdf, (Data Download Date : 18-08-2014).

14. www.ling.ohio-state.edu/~kbaker/pubs/ Singular_Value_Decomposition_ Tutorial.pdf, (Data Download Date : 18-08-2014).

15. www.ling.ohio-state.edu/~kbaker/pubs/ Singular_Value_Decomposition_ Tutorial.pdf, (Data Download Date : 18-08-2014).

16. www.ling.ohio-state.edu/~kbaker/pubs/ Singular_Value_Decomposition_ Tutorial.pdf, (Data Download Date : 18-08-2014).

17. www.ling.ohio-state.edu/~kbaker/pubs/ Singular_Value_Decomposition_ Tutorial.pdf, (Data Download Date : 18-08-2014).

18. http://see.stanford.edu/materials/lsoeldsee263/16-svd.pdf, (Data Download Date : 18-08-2014).

19. http://home.bi.no/a0710194/Teaching/BI-Mathematics/GRA-6035/2010/ lecture2.pdf, (Data Download Date : 18-08-2014).

20. http://see.stanford.edu/materials/lsoeldsee263/16-svd.pdf, (Data Download Date : 18-08-2014).

21. **Thomas K. L., Peter W. F., Darrell L., (1998),** "*An Introduction to Latent Semantic Analysis", *Discourse Processes, vol. 25, pp.2.

22. **Scott D., Susan T. D., George W. F., Thomas K. L., Richard H., (1998),** "*Indexing by Latent Semantic Analysis*", Journal of the American Society for Information Science, vol. 41, pp.391.

23. **Scott D., Susan T. D., George W. F., Thomas K. L., Richard H., (1998),** "*Indexing by Latent Semantic Analysis",* Journal of the American Society for Information Science, vol. 41, pp.391.

APPENDICES A

CURRICULUM VITAE

PERSONAL INFORMATION
Surname, Name: Pojon, Murat
Date and Place of Birth: 16 June 1985, Ankara
Marital Status: Single
Phone: +90 537 363 94 83
Email: muratpojon@gmail.com

EDUCATION

Degree	**Institution**	**Year of Graduation**
M.Sc.	Çankaya Univ., Computer Engineering	2014
B.Sc.	METU, Mathematics	2009
High School	METU High School	2002

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2013- Present	Arkadaş Publishing	Software Developer

FOREIGN LANGUAGES

Advanced English