**Doğal Dil İşlemede Temel Semantik Analiz**

**Semantik** kelimelerin, işaretlerin ve cümle yapılarının yorumlanmasıdır. Örneğin farklı dillerde Farklı dillerde “ Hallo, Namaste, Salut, Hello “ kelimelerinin hepsi merhaba anlamına gelmektedir.

Bir dilde iki tip semantik (anlam) vardır.

1. **Kavramsal Anlam (Sözlük Anlamı)** : Needle ( İğne) : ucu sivri çelik alet. Örnek: elime iğne battı
2. **İlişkisel (Mecazi) Anlam** : Needle ( Painful) : iğneleyici Örnek : İğneleyici konuştu

**İçerik Analizi ( Content Analysis) :** İçerik analizi, keşfetmek, ortaya çıkarmak veya cevaplamak için bir medya koleksiyonunu incelemeye yönelik resmi bir metodolojidir.

İki şekilde yapılıyor

1. Niceliksel (Quantitative): Terim frekansları, stop wordler
2. Niteliksel (Qualitative):

Semantik analiz kelime bazlı, cümle yada kısa metin bazlı veya döküman bazlı yapıldığı görülmektedir.

**Vector Space Model :**  Döküman, metin, cümle yada kelimeleri yüksek boyutlu bir uzayda temsil etmenin bir yöntemi

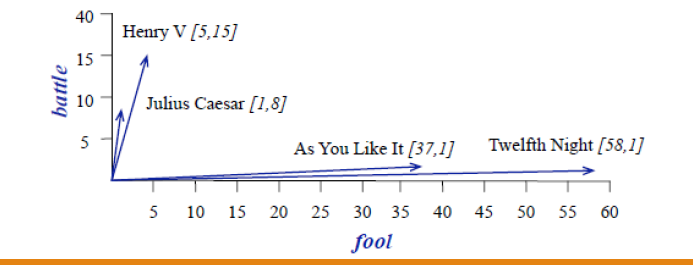
**Term-Document Matrix:** Aşağıda görülen matris terim döküman matrisidir. Shekspir’in 4 farklı oyununaait kelimelerin dağılımı görülmektedir.



Her oyun, sütun vektörü olarak (1x4) görülebilir (kırmızı seçimler)

***“Dökümanlar için şunu söyleyebiliriz :”Benzer dökümanlar benzer vektörlere sahiptir çünkü benzer dökümanlar benzer kelimeleri bulundurma eğilimindedir. “***

Aşağıda vector space modeli sadece basit şekilde gösterim açısından 2 kelime kullanılarak **battle** ve **fool** kelimeleri açısından bu oyunların vector space model üzerinde gösterimi yer almaktadır.



Şekilde Henry V oyununun [5, 15] lik bir vektörle temsil edildiği görülmektedir.

**Word Vector**

Aynı matris üzerinde Word Vector elde edilebilir.

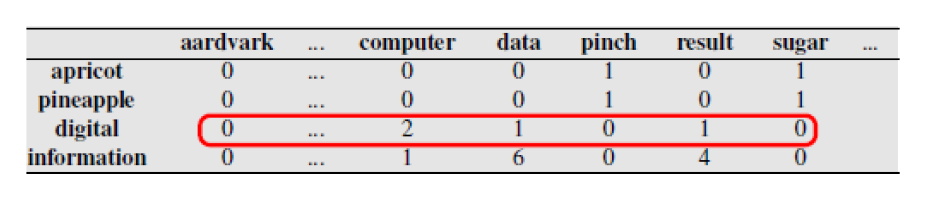
Fool kelimesi 4 oyun metni için [37,58,1,5] şeklinde temsil edilir.

***“Benzer kelimeler benzer vektörlere sahiptir çünkü benzer dökümanlarda olma eğilimindedir. “***

**Word to Word Matrix**

Bu matristeki hücreler bu dökümanda satırda yer alan kelimenin sütunda yer alan kelime ile kaç

defa birlikte geçtiğini gösteriyor. apricot Genelde satırda yer alan kelimenin geçtiği metinde bulunarak 4 karakter solunda ve sağında sütundaki kelimenin kaç defa geçtiğine bakılarak elde edilebilir.



Yukarıdaki tabloya göre verilen bir dökümanda word-word matrix oluşturulabilir.

Her bir kelimenin diğer kelimelerle birlikte geçme sayıları matrisin hücrelerine yerleştirilir. ( -4 ----- kelime ----- +4 bu yöntem kullanıldığında bir kelimenin diğer kelime ile 1,2,3,4 kelime mesafede olsa bile hücreye sayısal olarak yansıtılmaktadır. Yani eş oluşuma bir de mesafe eklenmiştir. Örnek digital ve computer kelimeleri birbirlerine bitişik yada 4 kelime mesafede olsa bile hesaplama yapılırken +1 olarak dikkate alınır ve ilgili hücreye metin içerisindeki toplam değer yazılır)

Aşağıdaki şekilde “Digital” kelimesinin “resut” kelimesi ile 1 defa , data kelimesi ile 1 defa birlikte geçtiği anlaşılmaktadır.

****

information(6,4,2,2,2,3)

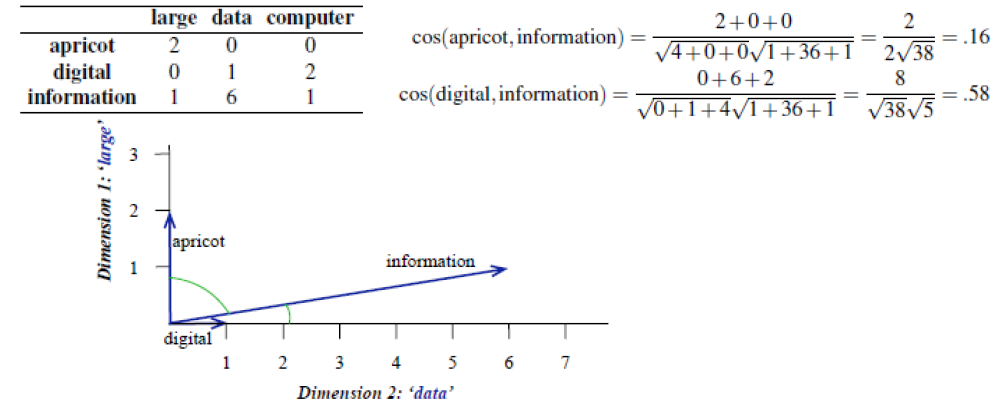
Computer

***“Genel olarak, pencere ne kadar kısa olursa temsiller o kadar sözdizimsel olur, çünkü bilgi hemen yakındaki kelimelerden gelir; pencere ne kadar uzun olursa ilişkiler o kadar anlamsal olur.”***

**Benzerliği Hesaplama**

İki hedef sözcük v ve w arasındaki benzerliği tanımlamak için ve vektör benzerliği ölçüsü vermek için bir ölçüme ihtiyacımız var.

En yaygın benzerlik ölçüsü, vektörler arasındaki açının **kosinüsüdür.**



**Anlamsal Sınıflandırma (Semantıc Classification)**

Belgeleri sınıflandırmak için temel yöntem, belge kelimelerinin, her bir konunun verilen anahtar kelime listesiyle karşılaştırılmasıdır.

Bir konudaki maksimum anahtar kelime sayısı, belgelerin konusunu belirleyebilir.

A, B, C

Araştırma Ödevi : **Word2Vec modeli nedir ?**

**https://medium.com/@muhammedbuyukkinaci/word2vec-nedir-t%C3%BCrk%C3%A7e-f0cfab20d3ae**

**Latent Semantic Analysis (LSA) :**

**Topic Modelling**

**Use-Cases of Topic Modeling**

**Simple applications in which this technique is used are documented clustering in text analysis, recommender systems, and information retrieval. More detailed use-cases of topic modeling are:**

**Resume Summarization:** It can help recruiters to evaluate resumes by a quick glance. They can reduce effort in filtering pile of resume.

**Search Engine Optimization**: online articles, blogs, and documents can be tag easily by identifying the topics and associated keywords, which can improve optimize search results.

**Recommender System Optimization:** recommender systems act as an information filter and advisor according to the user profile and previous history. It can help us to discover unvisited relevant content based on past visits.

**Improving Customer Support:** Discovering relevant topics and associated keywords in customer complaints and feedback for examples product and service specifications, department, and branch details. Such information help company to directly rotated the complaint in respective department.

**In the healthcare industry,** topic modeling can help us to extract useful and valuable information from unstructured medical reports. This information can be used for patients treatment and medical science research purpose.

**Kaynak:** <https://www.youtube.com/watch?v=Fy0bF7u6W20>

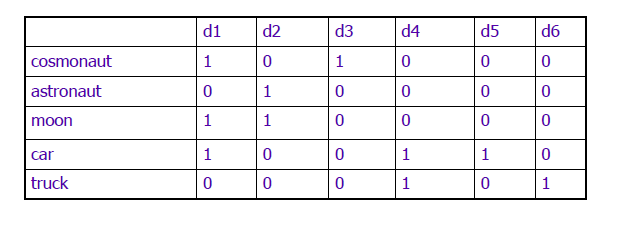
Denetimsiz öğrenme yöntemidir.

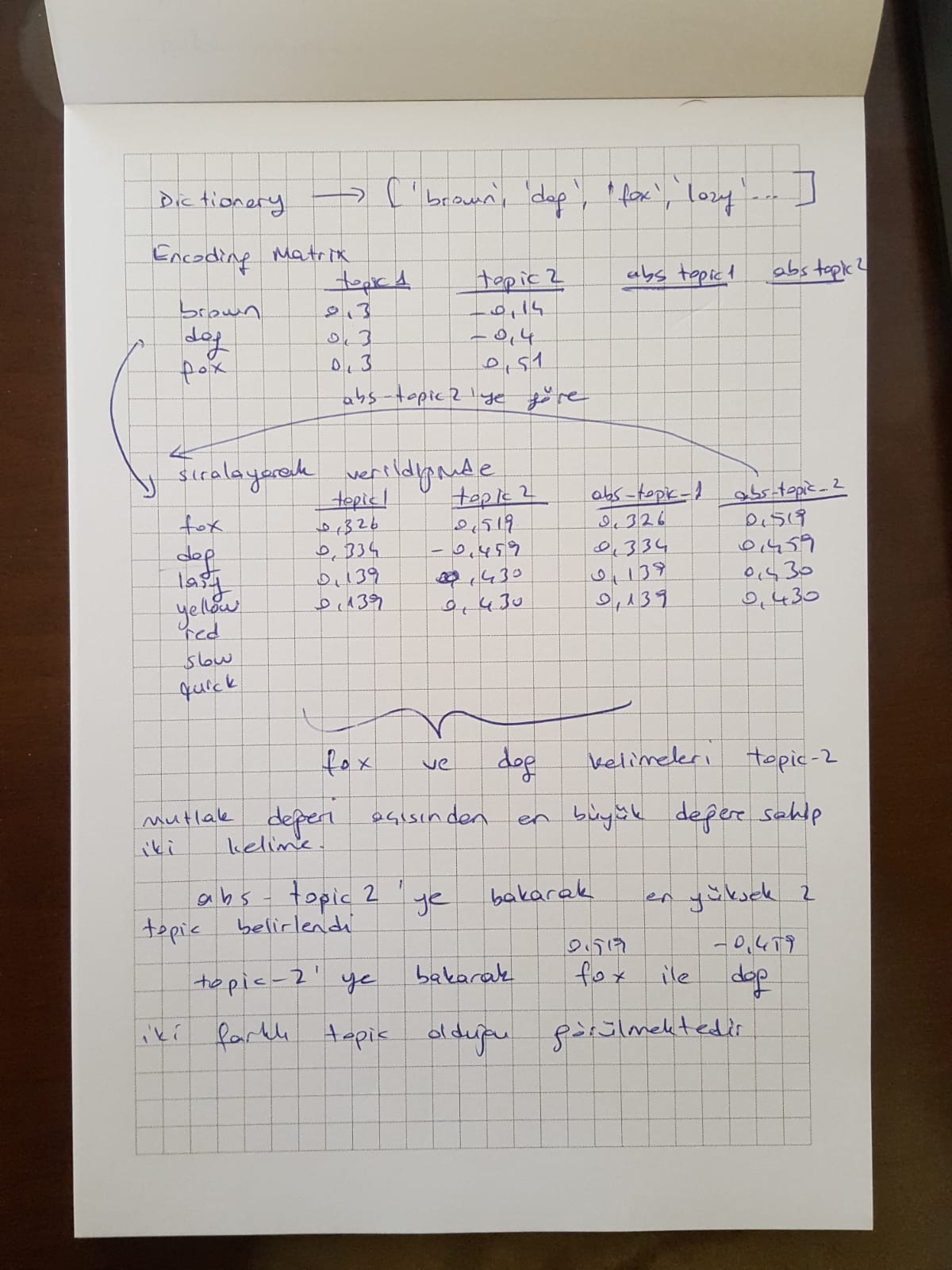
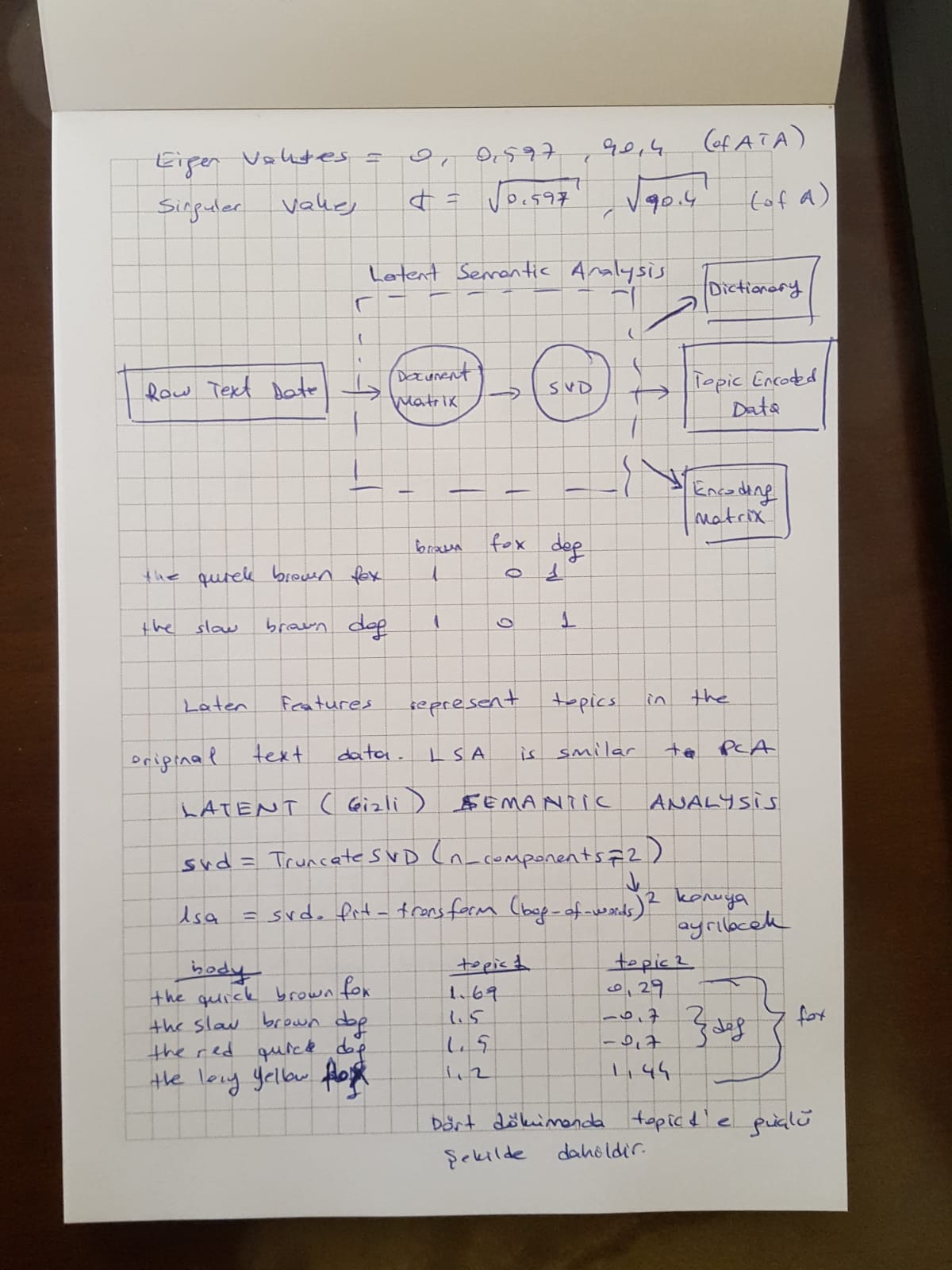
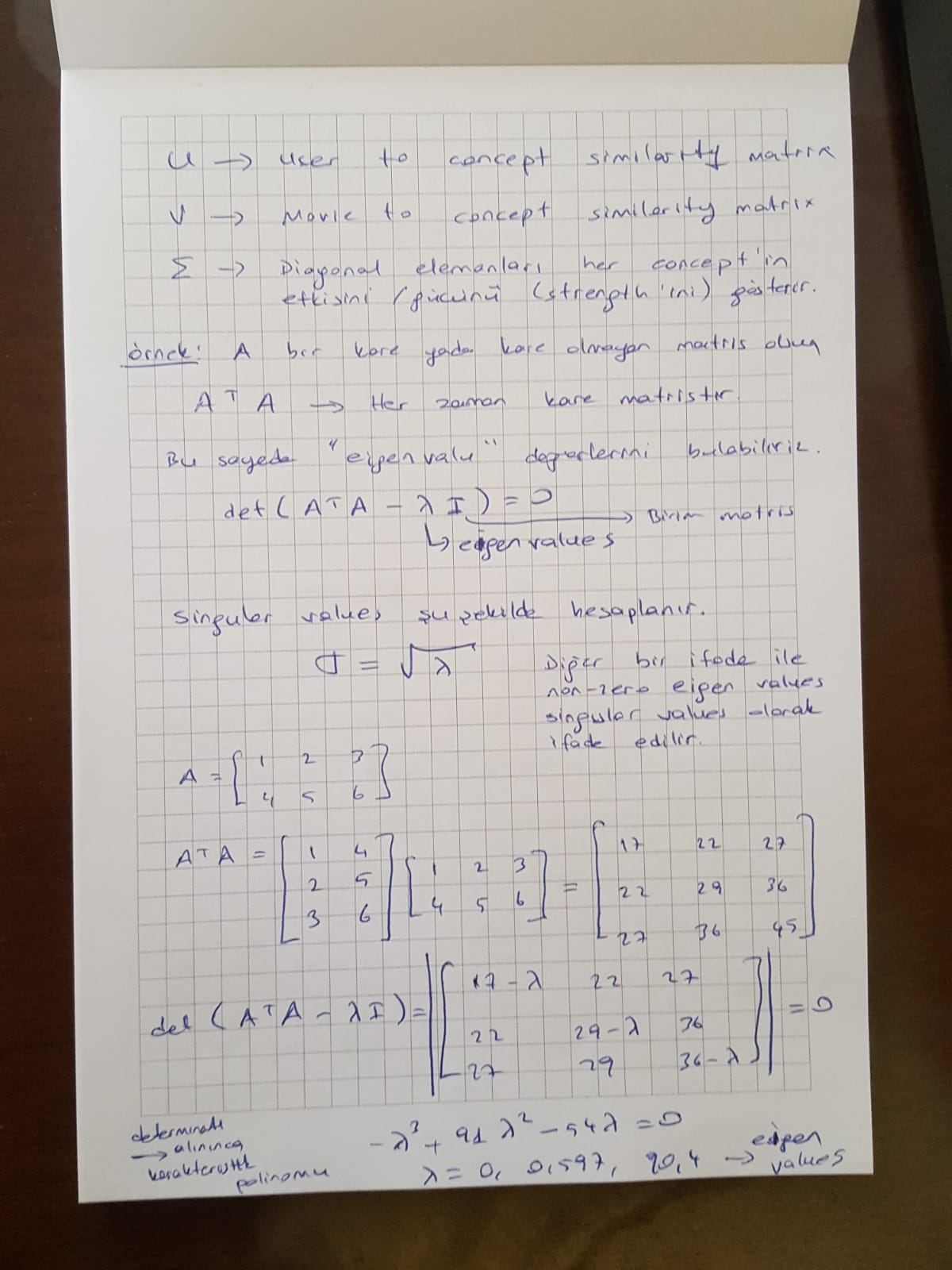
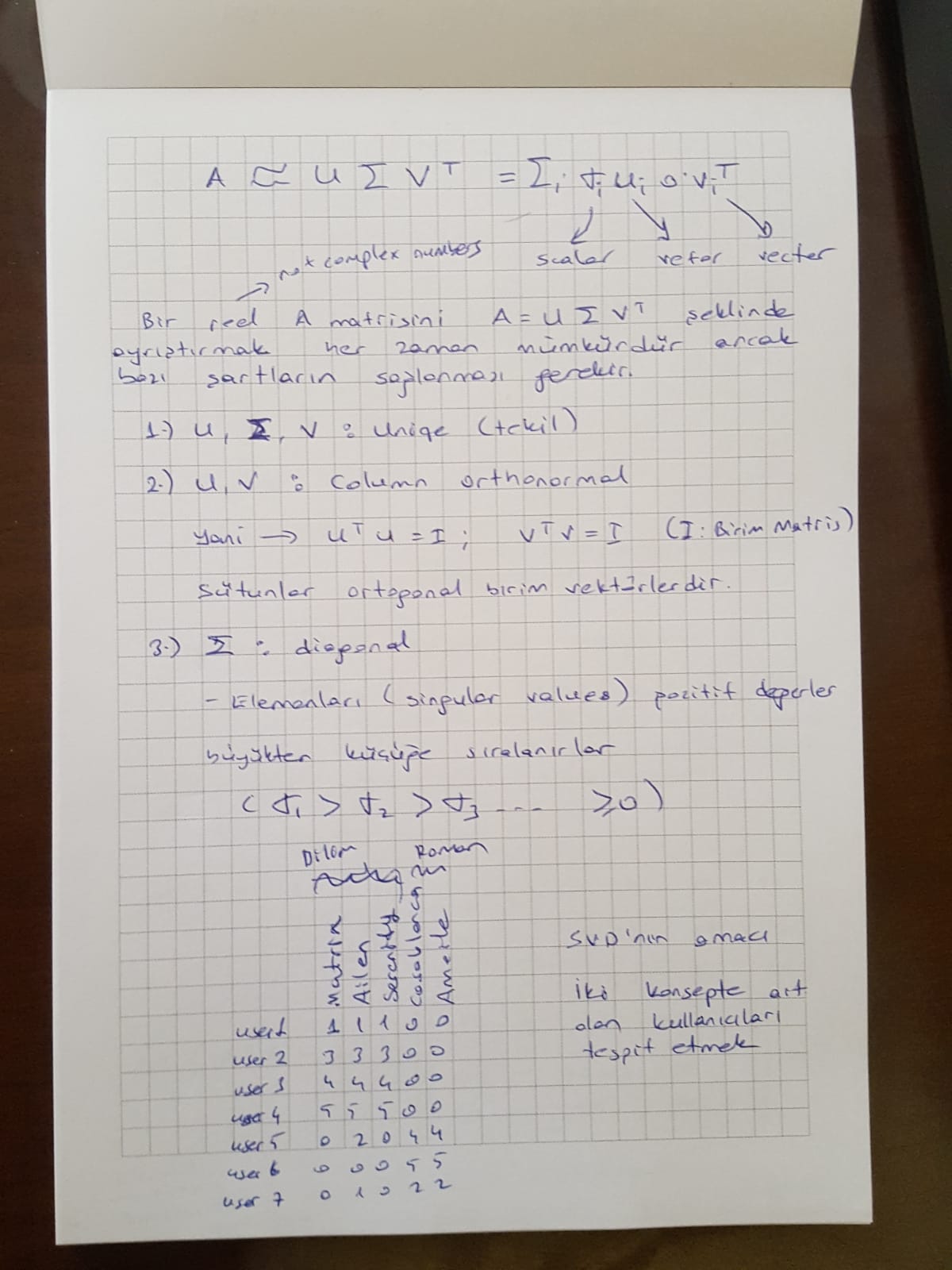
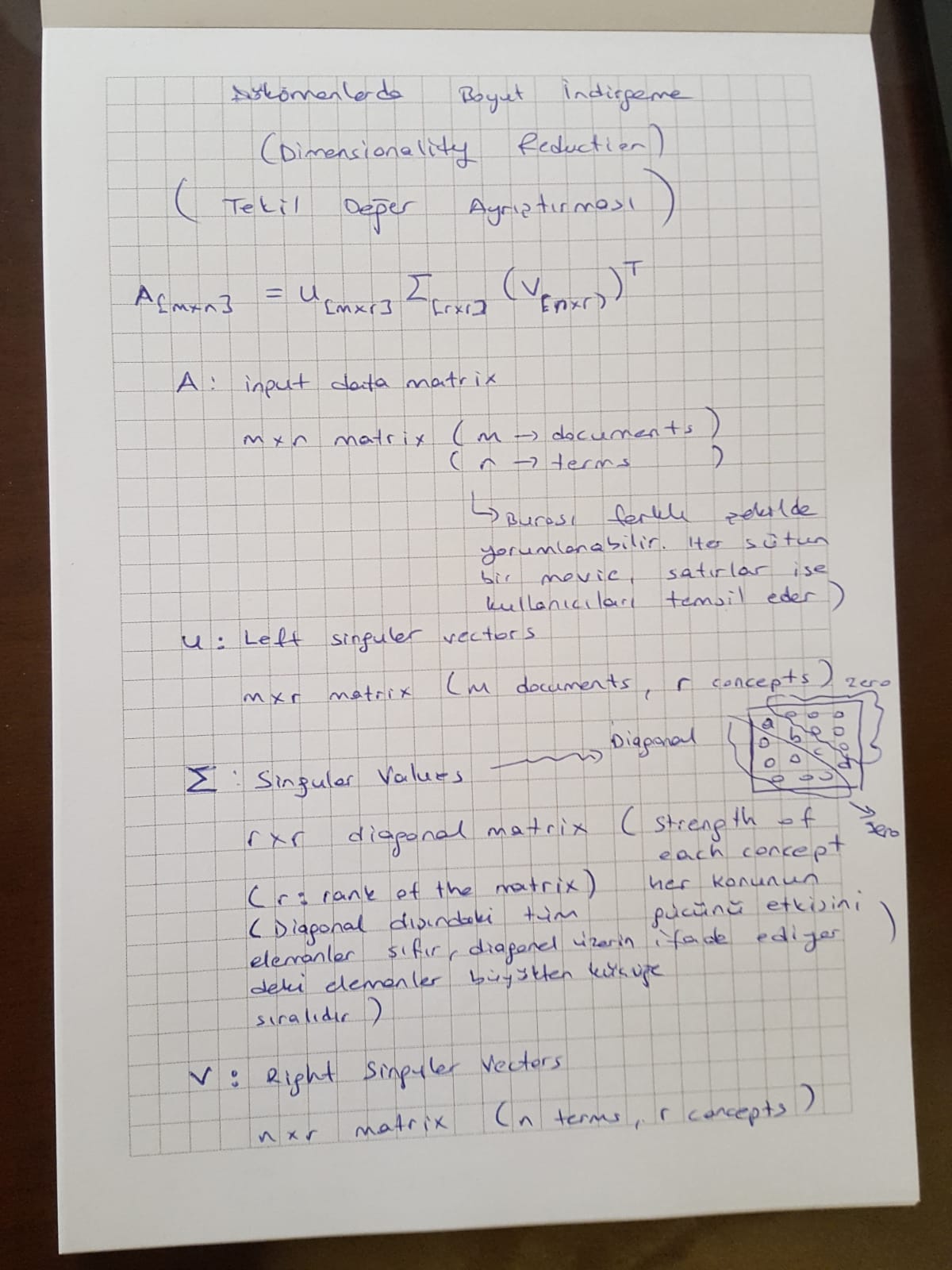
Girdi olarak metin verisi alınır. NLP çalışmalarında bir terim döküman matrisi elde edilir.

Aşağıdaki tabloda d1,d2,... d6 dökümanları satırlada ise kelimeler yer almaktadır.

Aşağıdaki matris 5x6 ‘lık bir döküman matrisini göstermektedir. Bu matrisin **SVD** (Singular Value Decomposition) yöntemi ile tekil bileşenler analizi yapılarak **Topic\_Encoded\_Data** elde edilir. Topic encoded data, her bir dökümanın algoritma çalıştıktan sonra kaç topic için (küme sayısı gibi düşünebilirsiniz) ayrılacağını göstermektedir. Topiclerin neler olduğu ise **Dictionary Encoded Data** kullanılarak elde edilir.

**Row Text Data 🡪 Document Matrix 🡪SVD 🡪 Topic Encoded Data, Dictionary Encoded Data**





**Latent Semantic Analysis: How does it work, and what is it good for?**

[**Genevieve Gorrell**](http://www.dcs.shef.ac.uk/~genevieve/)**, May 2005. Last updated January 2007.**

**1 Introduction**

Briefly, [Latent Semantic Analysis](http://lsa.colorado.edu/) is a method in which [singular value decomposition](http://mathworld.wolfram.com/SingularValueDecomposition.html) is used to form semantic generalisations from textual passages. It uses the fact that certain words appear in similar contexts to establish relationships between the meanings of the words. It allows textual passages to be compared to each other more intelligently than by directly comparing the words they share. Words that never appear together can be meaningfully compared etc.

This tutorial explains how LSA works from a technical perspective. It assumes no knowledge of singular value decomposition, and minimal maths background. It is divided into three sections. The first explains singular value decomposition. The second describes LSA. The third discusses the meaning of LSA and gives some context.

**2 Singular value decomposition**

**2.1 Eigen Decomposition**

Let us begin with [eigen decomposition](http://mathworld.wolfram.com/EigenDecomposition.html). Imagine you have a square, diagonally-symmetrical matrix, that is to say, if you flipped it on an axis from its top left to its bottom right corners, it would look just the same. (It is equal to its own transpose.) We are starting with this simple case, and we can then work up to more complicated cases. Below is an example square diagonally-symmetrical matrix. The matrix happens also to be diagonal, that is to say, all off-diagonal elements (those in which row and column number are not equal) are set to zero, but this fact is not relevant. The example is chosen because it makes the decomposition process transparent and easy to follow.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [ | 3 | 0 | 0 | ] |
| 0 | 2 | 0 |
| 0 | 0 | 1 |

For any matrix of this form, there will be an [orthonormal basis](http://mathworld.wolfram.com/OrthonormalBasis.html) set (a set of normalised vectors that are all perpendicular to each other) in which the matrix can be described purely in terms of its "strength" in each direction. This is rather like a [Fourier transform](http://mathworld.wolfram.com/FourierTransform.html), in which a sound wave can be described purely in terms of its amplitude in each of a number of frequencies. A more accessible, though less strictly accurate, analogy would be to consider trying to describe a colour entirely in terms of quantities of red, blue and green light. So the eigen decomposition of a matrix is the set of vectors that form the characteristic orthonormal basis set for this matrix, and for each vector, an associated strength, known as the eigenvalue. Below is the eigen decomposition of the above matrix.

|  |  |  |
| --- | --- | --- |
|  | [1,0,0] | 3 |
|  | [0,1,0] | 2 |
|  | [0,0,1] | 1 |

The three eigenvectors, when multiplied by the original matrix, become scaled versions of themselves; they do not change direction. (Multiply a column vector by a matrix to obtain a row vector by dot-producting the vector with each column of the matrix in turn. Dot product two vectors together, which must have the same length, by multiplying each number in the first with its positional counterpart in the second and then summing.) For any square symmetrical matrix there will be a set of vectors that have this property, and the number of such vectors is referred to as the ``rank'' of the matrix. The rank of the matrix will be no greater than the number of rows (or columns). Between them, the vectors, each with associated scaling factor, completely describe the matrix; the matrix is reobtainable from its decomposition without any loss of information. The eigen decomposition of a matrix M is often described using the following equation;  
  
λ v = M v  
  
in which λ is the eigenvalue, v is the eigenvector and M is the matrix. The equations simply says that multiplying the eigenvector by the matrix has the same effect as scaling it by the eigenvalue.

To get back the original matrix from this set of vectors and values, you would outer product each vector with itself to form a square diagonally-symmetrical matrix, multiply each matrix by the appropriate value and add all the matrices up together. ("Outer product" is a particular kind of vector multiplication in which two vectors are multiplied together to form a matrix. The number at row 1, column 1 is the product of the first number in each vector, the number at row 1, column 2 is the product of the first number in the first vector and the second number in the second, and the number at row 2, column 1 is the product of the second number in the first vector with the first number in the second vector etc. It is clear therefore that outer producting a vector with itself would form a square, diagonally-symmetrical matrix. Matrix addition simply involves adding each number to its positional counterpart in the other matrix, and therefore requires that the two matrices have indentical dimensionality, which in this case they will, since the eigenvectors have identical dimensionality.)

If the eigenvectors were to be presented as a matrix P, and the values as a diagonal matrix D, then the decomposition could be presented as follows, where M is the original matrix and P(T) is the transpose of matrix P:

M = P D P(T)

So in the example above:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [ | 1 | 0 | 0 | ] |  | [ | 1 | 0 | 0 | ] |  | [ | 3 | 0 | 0 | ] |
| P = | 0 | 1 | 0 | P(T) = | 0 | 1 | 0 | D = | 0 | 2 | 0 |
|  | 0 | 0 | 1 |  | 0 | 0 | 1 |  | 0 | 0 | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | [ | 1 | 0 | 0 | ] |  | [ | 3 | 0 | 0 | ] |  | [ | 1 | 0 | 0 | ] |
| M = | 0 | 1 | 0 | \* | 0 | 2 | 0 | \* | 0 | 1 | 0 |
|  | 0 | 0 | 1 |  | 0 | 0 | 1 |  | 0 | 0 | 1 |

To multiply two matrices together, dot product the first row of the first with the first column of the second to get the number at row 1, column 1, dot product the first row of the first with the second column of the second to get the number at row 1, column 2, dot product the second row of the first with the first row of the second to get the number at row 2, column 1 etc. Dot product two vectors together (which must have the same length) by multiplying each number in the first with its positional equivalent in the second and then summing.

By convention, an eigen decomposition is shown in order of descending eigenvalues. There are many different ways to perform an eigen decomposition, and the most efficient depends on the nature of the matrix. For example, a matrix which comprises mostly zeros (a "sparse" matrix) can be decomposed using techniques that exploit this fact. One simple method is to repeatedly multiply a vector by the matrix, until the vector settles on the largest eigenvector. Matlab also has a very quick and efficient implementation.

**2.2 Extension to rectangular matrices**

Thus far we have discussed only the decomposition of square, diagonally-symmetrical matrices. Extension to arbitrary matrices requires the inclusion of an additional vector set. The **singular value decomposition** of a matrix comprises a set of matched normalised vectors, each pair with its associated singular value. The matrix can be reacquired from its decomposition by outer-producting each vector pair together, multiplying by the singular value, and then adding all the resulting matrices up together. This is often described as follows;

M = U  V(T)

in which M is the matrix, U is the matrix of left singular vectors,  is the diagonal matrix of singular values and V(T) is the transpose of the matrix of right singular values. The relationship between the singular value decomposition of a matrix and the eigen decomposition is as follows: Given an arbitrary matrix M, the left singular vector set is equal to the eigen decomposition of the matrix formed by multiplying M by its transpose. (The transpose of a matrix is the matrix flipped on its top left to bottom right diagonal.) The right singular vector set is equal to the eigen decomposition of the matrix formed by multiplying the transpose of M by M. The singular values are simply the square roots of the eigen values of these matrices. The eigenvalues are the same for each. That the singular values would be the square roots of the eigenvalues in this case is intuitive given that the eigenvalues are calculated on what is effectively the square of M. The left singular vectors are equal in dimensionality to the length of M, and the right singular vectors are equal in dimensionality to the width of M.

**2.3 Useful properties of eigen and singular value decompositions**

Recall that the eigen and singular value decompositions of a matrix are by convention given in decreasing order of eigen (or singular) value, and that the values indicate the "amount" of that vector (pair) present in the matrix, or the impact that (pair of) direction(s) has on the behaviour of the matrix. By taking the top n vectors, those with the largest values, we can describe a least squared error approximation to the original matrix using a smaller set of numbers: we have discarded those which had less impact on the matrix anyway. A matrix thus reduced has some useful properties. Firstly, it can constitute a compression (depending on the nature of the matrix). Secondly, the discarding of the details can act as noise reduction, or removal of detail or richness unnecessary to the task, and can lead to an improvement in performance, depending on context. It is the second of these properties that makes LSA possible.

**3 Latent Semantic Analysis**

Latent Semantic Analysis (LSA) describes the application of SVD to a particular kind of matrix. Imagine you have a set of textual passages:

"The man walked the dog"  
"The man took the dog to the park"  
"The dog went to the park"

You prepare a matrix of word counts like so:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Passage 1 | Passage 2 | Passage 3 |
| the | 2 | 3 | 2 |
| man | 1 | 1 | 0 |
| walked | 1 | 0 | 0 |
| dog | 1 | 1 | 1 |
| took | 0 | 1 | 0 |
| to | 0 | 1 | 1 |
| park | 0 | 1 | 1 |
| went | 0 | 0 | 1 |

To make it explicit, this matrix M is the sum of the outer products of paired vectors; document and wordbag vectors. Each data item comprises the wordbag vector, positioning it in word space, and a trivial document vector comprising a single one in a vector of zeros, positioning it uniquely in document space. This document vector can be thought of as a placeholder, and the matrix of document vectors is in fact the identity matrix (comprising all zeros but for ones on the diagonal, resulting in a matrix that when multiplied by another does not modify it). This might seem an irrelevant complication at this point but it becomes important shortly.

We decompose the matrix M using SVD, to obtain two sets of normalised singular vectors, each with an associated singular value:

[ 0.46, 0.73, 0.51] [5.03] [ 0.82, 0.24, 0.09, 0.34, 0.14, 0.25, 0.25, 0.10]  
[-0.77, 0.04, 0.64] [1.57] [-0.10,-0.47,-0.49,-0.06, 0.02, 0.43, 0.43, 0.40]  
[ 0.45,-0.68, 0.58] [1.09] [-0.01,-0.22, 0.41, 0.31,-0.63,-0.10,-0.10, 0.53]

A corpus of three documents can produce no more than three singular triplets (pairs of vectors with associated singular value). However, a more realistic corpus might produce numbers of singular triplets in quadruple figures. Dimensionality reduction is then performed by discarding all but the top few hundred singular triplets. The precise number of singular triplets retained is chosen on an *ad hoc* basis. This stage will be simulated here by discarding the last singular triplet to produce a two-dimensional semantic space. Here are the remaining singular triplets;

[ 0.46, 0.73, 0.51] [5.03] [ 0.82, 0.24, 0.09, 0.34, 0.14, 0.25, 0.25, 0.10]  
[-0.77, 0.04, 0.64] [1.57] [-0.10,-0.47,-0.49,-0.06, 0.02, 0.43, 0.43, 0.40]

We use the U and V (left and right singular vector) matrices to rotate documents/wordbag vectors into a new space by multiplying the matrix of document vectors (the identity matrix, as discussed earlier) by the U matrix, or the matrix of wordbag vectors by the V matrix. We can then compare the semantic similarity of the documents using their dot products in this new space, which the LSA literature has demonstrated to be an improvement. Dot product provides a measure of similarity between two vectors, with 1 being a perfect match. We could equally well compare the words to each other. To compare documents to words, the matrix S would need to be taken into account, typically done by moving both documents and words through an additional sqrt(S) transform.

A typical use of LSA is, given a string such as a user query, to return the best match among a set of documents. This can be illustrated in the context of the above example. Suppose the query is "the dog walked". This string is used to form a vector in the same manner as the documents were; it becomes a ``pseudodocument''. The pseudodocument would therefore be;  
  
p = [1,0,1,1,0,0,0,0]

We can move this pseudodocument into the same space as the documents above, and compare them, by multiplying it by the inverse of the matrix of singular values and the matrix of right singular vectors;  
  
p' = inv(S) U(T) p =

|  |  |  |
| --- | --- | --- |
| [ | 0.25 | ] |
| 0.41 |

It can be seen that the pseudodocument is most similar to document 1. [Kevin Murphy](http://www.cs.ubc.ca/~murphyk/) has created a Matlab demonstration illustrating the above, and kindly allowed me to include it: [lsaDemoSmall2.m](https://staffwww.dcs.shef.ac.uk/people/G.Gorrell/lsaDemoSmall2.m)

Additionally, a number of techniques are available that allow the matrix to be preprocessed in such a way as to further increase the effectiveness of the technique. Words contribute to varying extents to the semantic profile of a passage; for example, the word ``the'' has little impact on the meaning of passages in which it appears. A word which distributes itself evenly among the documents in a collection is of little value in distinguishing between them. LSA can therefore be made more effective by reducing the impact of such words on word count matrix and increasing the impact of less evenly distributed words. Dumais (1990) outlines several methods of achieving this. The most sophisticated and effective of these will be presented here.  
  
c(ij) is the cell at column *i*, row *j* of the corpus matrix C. The entropy normalisation step involves modifying this value as follows;

Where gw(i) is the ``global weighting'' of the word at i, n is the number of documents in the collection, tf is the term frequency, ie. the original cell count, and gf is the global frequency, ie. the total count for that word across all documents.

**4 Context**

LSA aims to create a semantic space in which words and documents can be compared. The dimensionality of the semantic space is lower than the number of unique words in the document collection, which means that the space is less rich, and word vectors that in a larger space would have no projection on each other now do so. Documents that share no words would have no projection on each other in a space where each word was allocated its own dimension, but in the semantic space created through LSA, they do, which means that they can be compared.

Another way of looking at the problem of meaningful dimensionality reduction in the context of natural language understanding and LSA-style tasks would be to think in terms of defining n semantic objects, where n is chosen according to the task being performed, and treating words as unreliable indicators of these concepts. This problem can then be approached as a probability problem. This is the approach taken by Probabilistic LSA, in which Expectation Maximisation is used to solve the problem. PLSA is proving to be a very successful alternative to LSA, though depending on the context, each approach has its own advantages.

**Matlab Code Example:**

function lsaDemoSmall2()

% By Kevin Murphy

% Based on a tutorial by Genevieve Gorrell

% http://www.dcs.shef.ac.uk/~genevieve/lsa\_tutorial.htm

% See also http://en.wikipedia.org/wiki/Latent\_semantic\_analysis

sentence{1} = 'the man walked the dog.';

sentence{2} = 'the man took the dog to the park';

sentence{3} = 'the dog went to the park';

N = length(sentence);

words = [];

for i=1:N

w = regexp(sentence{i}, '(\w+)', 'match');

words = [words w];

end

words = unique(words);

% to facilitate comparison with the tutorial, we sort the words as follows

words = {'the', 'man', 'walked', 'dog', 'took', 'to', 'park', 'went'};

% Now build term-document matrix

W = length(words);

X = zeros(W, N);

for i=1:N

X(:,i) = X(:,i) + sentenceToWordCount(sentence{i}, words);

end

% truncated SVD

K = 2;

[U, S, V] = svds(X, 2);

% add test data to word-document matrix

query = 'the dog walked';

x = sentenceToWordCount(query, words);

%x = x/norm(x);

X(:,N+1) = x;

% embed all data into low dim space and plot

Z = inv(S)\*U'\*X;

figure(1);clf;

ndoc = size(Z,2);

for i=1:ndoc

plot(Z(1,i), Z(2,i), 'o');

hold on

h=text(Z(1,i), Z(2,i), sprintf('%d', i)); set(h,'fontsize',15);

line([0 Z(1,i)], [0 Z(2,i)]);

end

% compute pairwise similarities using dot product

%simMat = Z'\*Z;

% compute pairwise similarities using cosine measure

for i=1:ndoc

for j=i+1:ndoc

simMat2(i,j) = abs(Z(:,i)'\*Z(:,j) / (norm(Z(:,i)) \* norm(Z(:,j))));

end

end

%%%%%%%%%

function v = sentenceToWordCount(s, words)

tokens = regexp(s, '(\w+)', 'match');

v = zeros(length(words),1);

for j=1:length(tokens)

k = strmatch(tokens{j}, words, 'exact');

v(k) = v(k) + 1;

end