SBE 4032: Advanced Topics in Medical Informatics (2)

موضوعات متقدمة في المعلوماتية الطبية (2)

Lecture #3: Dimensionality Reduction — Linear Methods

Hisham Abdeltawab, Ph.D.
Assistant Professor,
Cairo University

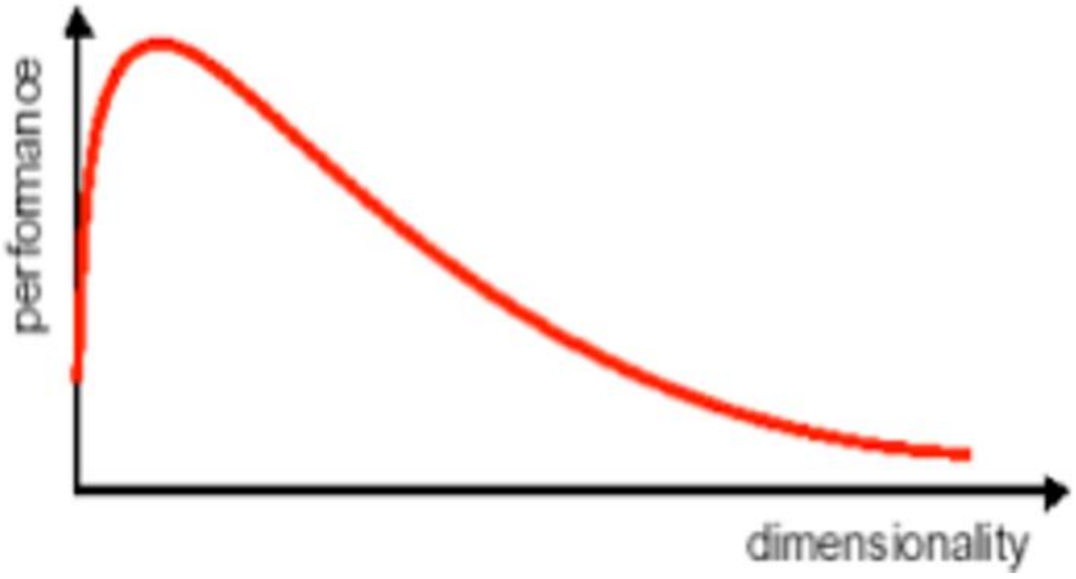
Why Dimensionality Reduction?

Features Matrix

p features (dimensionality)

n observations

0.6	0.3	….	….	….	…	…	0.2



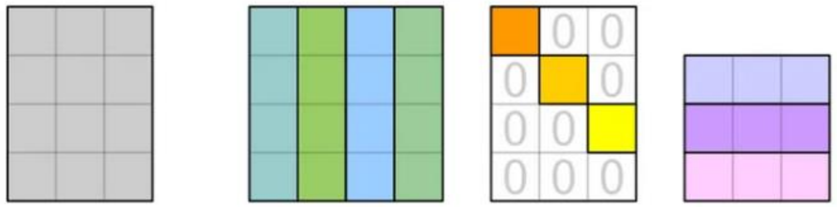
- It increases the performance machine learning models by removing noise and redundancy.

Singular Value Decomposition — SVD

- Applications of SVD:
  1. Dimensionality Reduction
     SVD allows for dimensionality reduction by retaining only the most significant singular values and vectors.
  2. Data Compression
     SVD is used in data compression tasks, reducing the storage requirements of a matrix.
  3. 3. Noise Reduction
     By using only the most significant singular values, SVD can help reduce the impact of noise in the data.

Singular Value Decomposition — SVD

For an arbitrary matrix; i.e., the matrix does not have to be square. The SVD of **X** is given by:



$$X = UDV^T$$

V

Data Matrix
Size = n x p

- Columns are called Left Singular Vectors and are orthonormal
- Columns are Eigen vectors of XX^T
- Size = n x n

- A diagonal matrix contains the singular values along the diagonal
- Singular values are $\sqrt{eigen\ values\ of\ X^TX}$
- Size = n x p

- Columns are called Right Singular Vectors and are orthonormal
- Columns are Eigen vectors of X^TX
- Size = p x p

Singular Value Decomposition — SVD

An approximation to the original matrix X is obtained via:

$$X_k = U_k D_k V_k^T$$

V_k

Approximation to the original Data Matrix
Size = n x p
Rank = k

- Contains the first k columns of matrix U
- Size = n x k

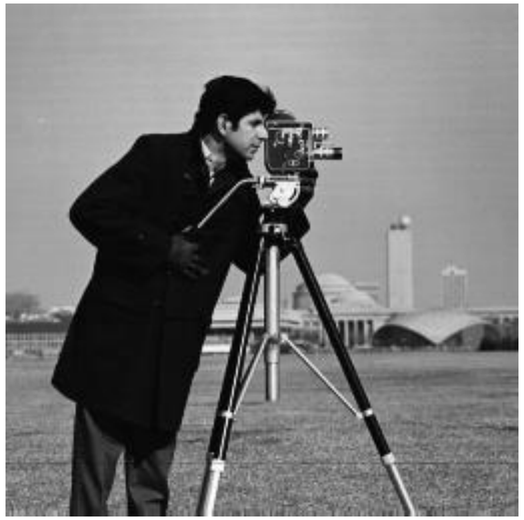
- Diagonal matrix whose diagonal elements are the k largest singular values of X.
- Size = k x k

- Contains the first k columns of matrix V
- Size = p x k

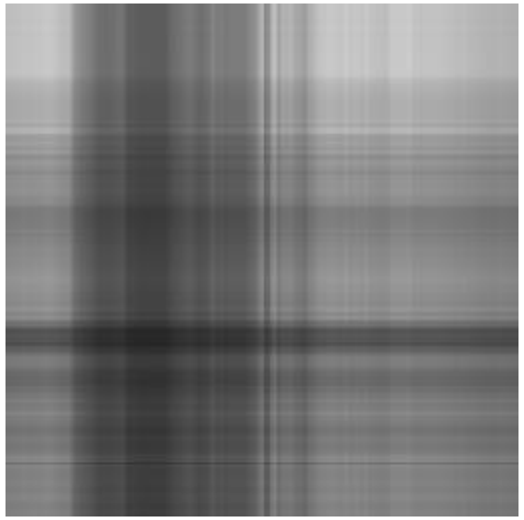
What is the rank of a matrix?

SVD: reconstruct an image by retaining only the highest singular values (image compression).

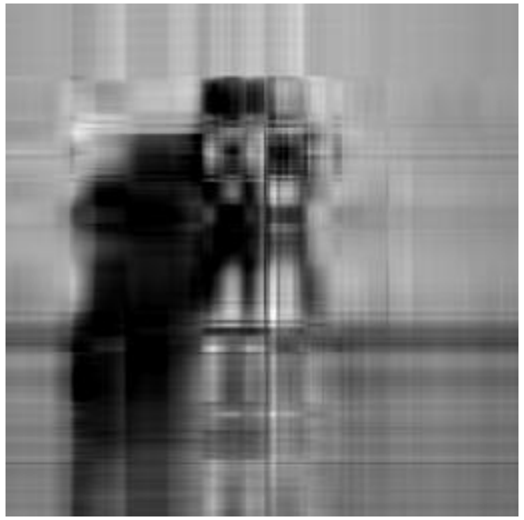
Original Image 256 x 256



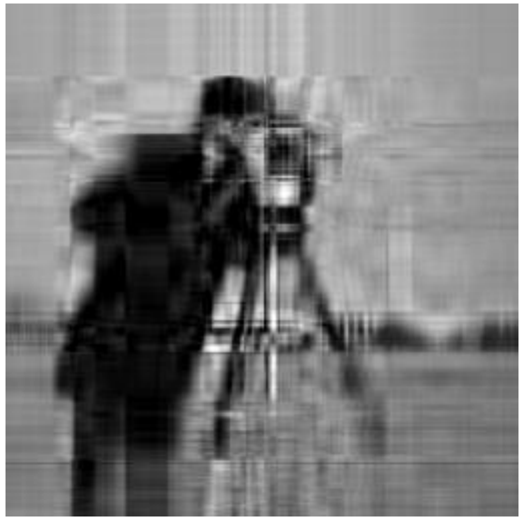
Reconstruction: Largest Singular Value



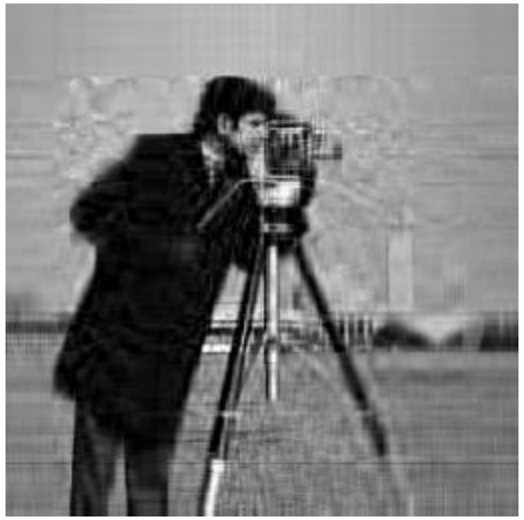
Reconstruction: Largest 5 Singular Values



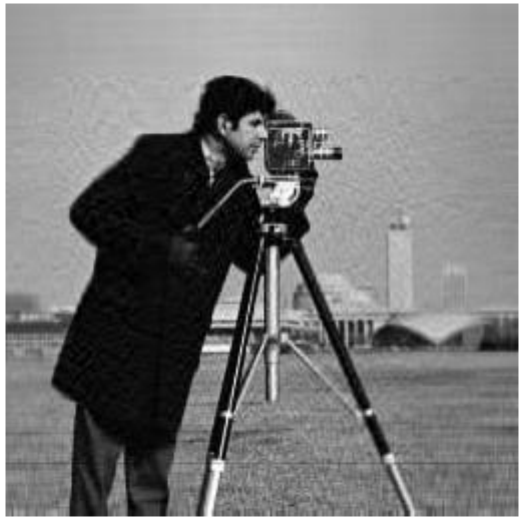
Reconstruction: Largest 10 Singular Values



Reconstruction: Largest 20 Singular Values



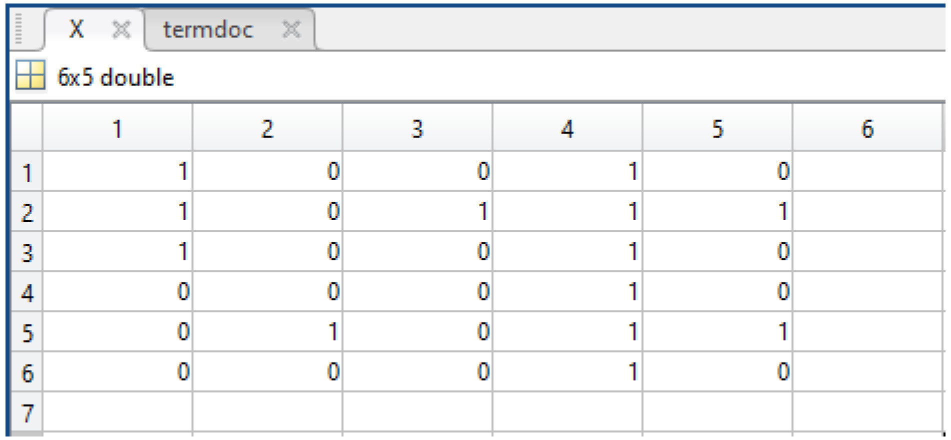
Reconstruction: Largest 50 Singular Values



Example: SVD applied to information retrieval (IR)

The documents in the corpus comprise a small set of book titles, and a subset of words have been used in the analysis, where some have been replaced by their root words (e.g., *bake* and *baking* are both *bake*).

Number	Title
Doc 1	How to *Bake Bread* Without *Recipes*
Doc 2	The Classic Art of Viennese *Pastry*
Doc 3	Numerical *Recipes*: The Art of Scientific Computing
Doc 4	*Breads*, *Pastries*, *Pies* and *Cakes*: Quantity *Baking Recipes*
Doc 5	*Pastry*: A Book of Best French *Recipes*
Term 1	bak (e, ing)
Term 2	recipes
Term 3	bread
Term 4	cake
Term 5	pastr (y, ies)
Term 6	pie


X termdoc

6x5 double

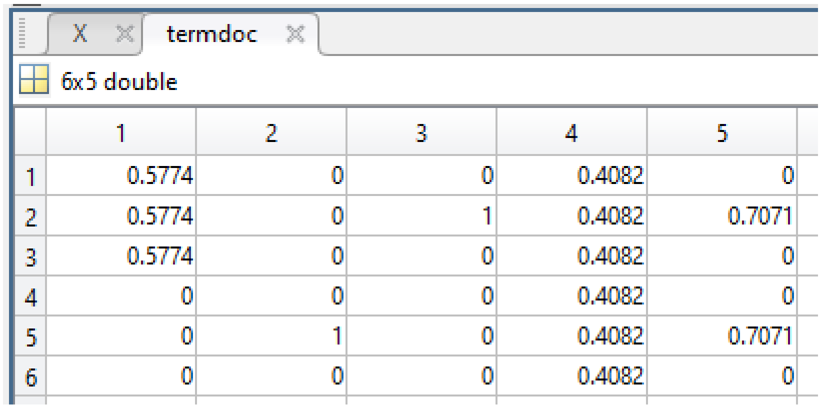
	1	2	3	4	5	6
1	1	0	0	1	0	
2	1	0	1	1	1	
3	1	0	0	1	0	
4	0	0	0	1	0	
5	0	1	0	1	1	
6	0	0	0	1	0	
7						

We start with a data matrix, where each row corresponds to a term, and each column corresponds to a document in the corpus. The elements of the term document matrix **X** denote the number of times the word appears in the document.

Example: SVD applied to information retrieval (IR)

```
load lsiex
% Loads up variables: X, termdoc, docs and words.
% Convert the matrix to one that has columns
% with a magnitude of 1.
[n,p] = size(termdoc);
for i = 1:p
     termdoc(:,i) = X(:,i)/norm(X(:,i));
end
```

We pre-process the matrix by normalizing each column such that the magnitude of the column is 1. This is done to ensure that relevance between the document and the query is not measured by the absolute count of the terms.



	1	2	3	4	5
1	0.5774	0	0	0.4082	0
2	0.5774	0	1	0.4082	0.7071
3	0.5774	0	0	0.4082	0
4	0	0	0	0.4082	0
5	0	1	0	0.4082	0.7071
6	0	0	0	0.4082	0

Say we want to find books about *baking bread*.

```
q1 = [1 0 1 0 0 0]';
```

If we are seeking books that pertain only to *baking*, then the query vector is:

```
q2 = [1 0 0 0 0 0]';
```

Example: SVD applied to information retrieval (IR)

- We can find the most relevant documents using the original term-document matrix by finding the cosine of the angle between the query vectors and the columns (i.e., the vectors representing documents or books)
- The higher cosine values indicate greater similarity between the query and the document.

```
% Find the cosine of the angle between
% columns of termdoc and a query vector.
% Note that the magnitude of q1 is not 1.
m1 = norm(q1);
cosq1a = q1'*termdoc/m1;
% The magnitude of q2 happens to be 1.
cosq2a = q2'*termdoc;
```

$$\cos\theta_{x,y} = \frac{\mathbf{x}^T\mathbf{y}}{\sqrt{\mathbf{x}^T\mathbf{x}}\sqrt{\mathbf{y}^T\mathbf{y}}}$$

The resulting cosine values are:

```
cosq1a = 0.8165, 0, 0, 0.5774, 0
cosq2a = 0.5774, 0, 0, 0.4082, 0
```

- If we use a cutoff value of 0.5, then the relevant books for our first query are the first and the fourth ones, which are those that describe *baking bread*.
- On the other hand, the second query matches with the first book, but misses the fourth one, which would be relevant.

Example: SVD applied to information retrieval (IR)

- Researchers in the area of IR have applied several techniques to alleviate the previous problem
- The idea is that some of the dimensions represented by the full term-document matrix are noise.

```
% Find the singular value decomposition.
[u,d,v] = svd(termdoc);
```

- We then find the representation of the query vector in the reduced space given by the first *k* columns of **U** in the following manner

$$\mathbf{q}_k = \mathbf{U}_k^T \mathbf{q}$$

Example: SVD applied to information retrieval (IR)

- The following code projects the query into the reduced space and also finds the cosine of the angle between the query vector and the columns.
- Researchers show that we do not have to form the full reduced matrix $\mathbf{X}_k$. Instead, we can use the columns of $\mathbf{V}_k$, saving storage space.

```
% Project the query vectors.
q1t = u(:,1:3)'*q1;
q2t = u(:,1:3)'*q2;
% Now find the cosine of the angle between the query
% vector and the columns of the reduced rank matrix,
% scaled by D.
for i = 1:5
    sj = d(1:3,1:3)*v(i,1:3)';
    m3 = norm(sj);
    cosq1b(i) = sj'*q1t/(m3*m1);
    cosq2b(i) = sj'*q2t/(m3);
end
```

```
cosq1b = 0.7327, -0.0469, 0.0330, 0.7161, -0.0097
cosq2b = 0.5181, -0.0332, 0.0233, 0.5064, -0.0069
```

Using a cutoff value of 0.5, we now correctly have documents 1 and 4 as being relevant to our queries on *baking bread* and *baking*.

Nonnegative Matrix Factorization (NMF)

- Motivation:
  - PCA and SVD might produce negative entries (in some situations, these entries are not easily interpreted as the entries in the original matrix.
  - In text-processing frameworks such as the previous example, the original matrix are zero or greater, such as representing counts.
  - Therefore, It would be desirable in these situations to have a method for reducing the dimensionality that is guaranteed to produce nonnegative features (interpretable).

- Nonnegative Matrix Factorization (NMF) casts matrix factorization as a constrained optimization problem that seeks to factor the original matrix into the product of two nonnegative matrices.

- Besides being easier to interpret, this type of matrix factorization has been shown to provide better results in information retrieval, clustering, and other applications

Nonnegative Matrix Factorization (NMF)

- The mathematical formalism of NMF for dimensionality reduction:
  - We find these factor matrices W and H by minimizing the following mean squared error objective function:

$$f(W,H) = \frac{1}{2}\,\|X - WH\|^2$$

$\|.\|$	X	W	H
Represents the Frobenius (Euclidian) norm: $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\|a_{ij}\|^2}$	Is the original data matrix with positive entries Size = n x p	Is nonnegative matrix Size = n x k K<p (k is defined by the user)	Is nonnegative matrix Size = k x p K<p (k is defined by the user)

It can not be solved analytically, and it is generally approximated numerically

Nonnegative Matrix Factorization (NMF)

Procedure – Multiplicative Update Algorithm

1. Initialize **W** as an $n \times k$ matrix with nonnegative entries that are randomly generated between zero and one.
2. Initialize **H** as a $k \times p$ random matrix with nonnegative entries between zero and one.
3. Update **H** using

$$\mathbf{H} = \mathbf{H} \ .\!*\ (\mathbf{W}^T\mathbf{X})\ ./\ (\mathbf{W}^T\mathbf{WH} + 10^{-9}).$$

4. Update **W** using

$$\mathbf{W} = \mathbf{W} \ .\!*\ (\mathbf{X}\,\mathbf{H}^T)\ ./\ (\mathbf{WHH}^T + 10^{-9}).$$

5. Repeat steps 3 and 4 until convergence or up to some maximum number of iterations.

Nonnegative Matrix Factorization (NMF)

- We can see from this algorithm that the results will be dependent on the initial random matrices.
- So, the analyst should obtain the factorization for different starting values and explore the results.
- The multiplicative update algorithm tends to be more sensitive to initialization than the alternating least squares approach, which is covered next.
- It has also been shown that the multiplicative update procedure is slow to converge.

Nonnegative Matrix Factorization (NMF)

Procedure – Alternating Least Squares

1. Initialize **W** as an $n \times k$ matrix with nonnegative entries that are randomly generated between zero and one.
2. Solve for **H** in the equation

$$\mathbf{W}^T\mathbf{W}\mathbf{H} = \mathbf{W}^T\mathbf{X}.$$

3. Set all negative elements in **H** to 0.
4. Solve for **W** in the equation

$$\mathbf{H}\mathbf{H}^T\mathbf{W}^T = \mathbf{H}\mathbf{X}^T.$$

5. Set all negative elements in **W** to 0.
6. Repeat steps 2 through 5 until convergence or up to some maximum number of iterations.

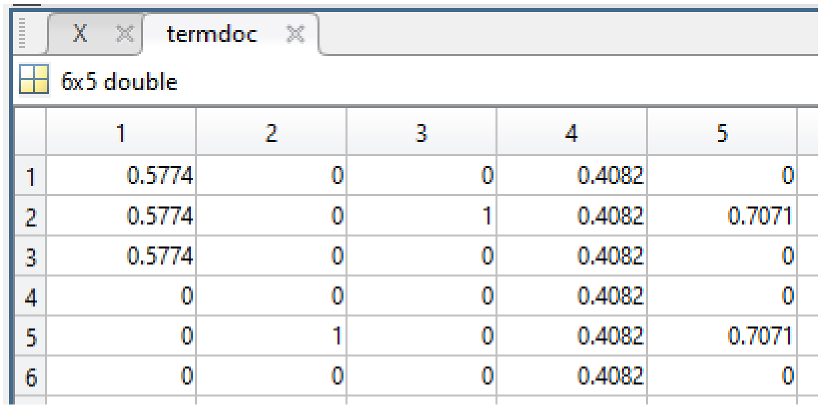
Nonnegative Matrix Factorization (NMF)

- Alternating Least Squares is an optimization algorithm commonly used in Non-Negative Matrix Factorization (NMF).
- It's an iterative algorithm that alternates between updating one factor matrix while holding the other fixed.
- The alternating optimization helps to iteratively minimize the objective function until convergence is achieved.

NMF: Example

```
load lsiex
% Loads up variables: X, termdoc, docs and words.
% Convert the matrix to one that has columns
% with a magnitude of 1.
[n,p] = size(termdoc);
for i = 1:p
     termdoc(:,i) = X(:,i)/norm(X(:,i));
end
```

We pre-process the matrix by normalizing each column such that the magnitude of the column is 1. This is done to ensure that relevance between the document and the query is not measured by the absolute count of the terms.



	1	2	3	4	5
1	0.5774	0	0	0.4082	0
2	0.5774	0	1	0.4082	0.7071
3	0.5774	0	0	0.4082	0
4	0	0	0	0.4082	0
5	0	1	0	0.4082	0.7071
6	0	0	0	0.4082	0

Apply NMF and set k=3

```
[W,H] = nnmf(termdoc,3,'algorithm','mult');
```

Recall that we had the following queries and are looking for documents that match them.

```
q1 = [1 0 1 0 0 0]';
q2 = [1 0 0 0 0 0]';
```

NMF: Example

Now we compute the rank *k* approximation to our term-document matrix that we obtained using nonnegative matrix factorization.

```
termdocnmfk = W * H;
```

We use the cosine measure and the columns of our approximated termdocument matrix to find the closest matches to our queries.

```
for i = 1:5
     m1 = norm(q1);
     m2 = norm(termdocnmfk(:,i));
     cosq1c(i) = (q1' * termdocnmfk(:,i))/(m1*m2);
     m1 = norm(q2);
     m2 = norm(termdocnmfk(:,i));
     cosq2c(i) = (q2' * termdocnmfk(:,i))/(m1*m2);
end
```

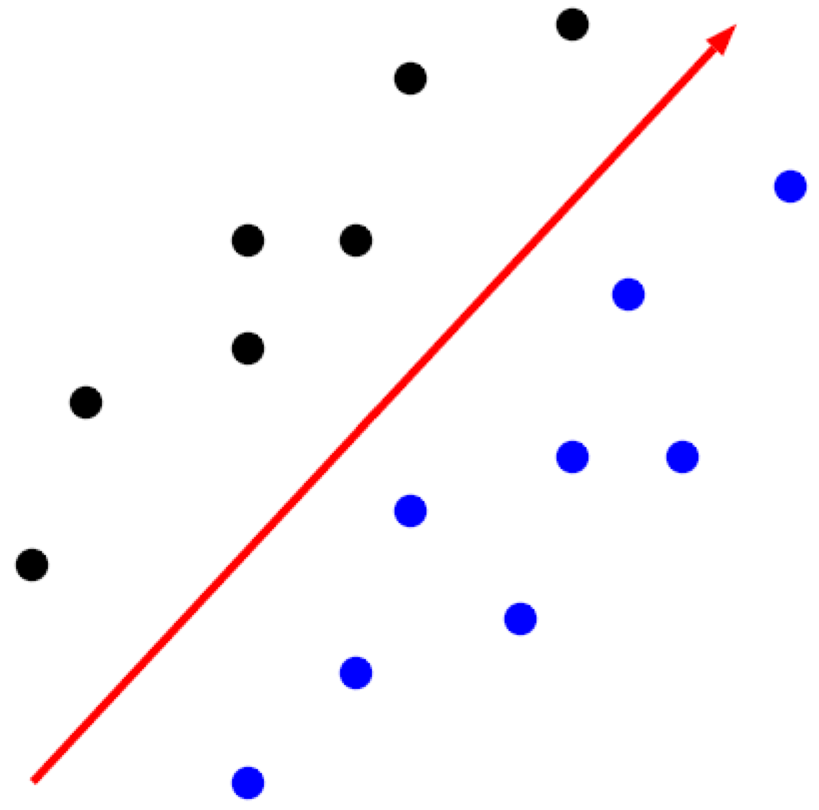
For this run, our results are

```
cosq1c = 0.7449 0.0000 0.0100 0.7185 0.0056
cosq2c = 0.5268 0.0000 0.0075 0.5080 0.0043
```

If we use a threshold of 0.5, then we see that the first and fourth documents match our queries.

Linear Discriminant Analysis (LDA)

- PCA aims to find the most accurate data representation in a lower dimensional space spanned by the maximum variance directions.
- However, such directions might not work well for tasks like classification.
- Here we present a new data reduction method that tries to preserve the discriminatory information between different classes of the data set.

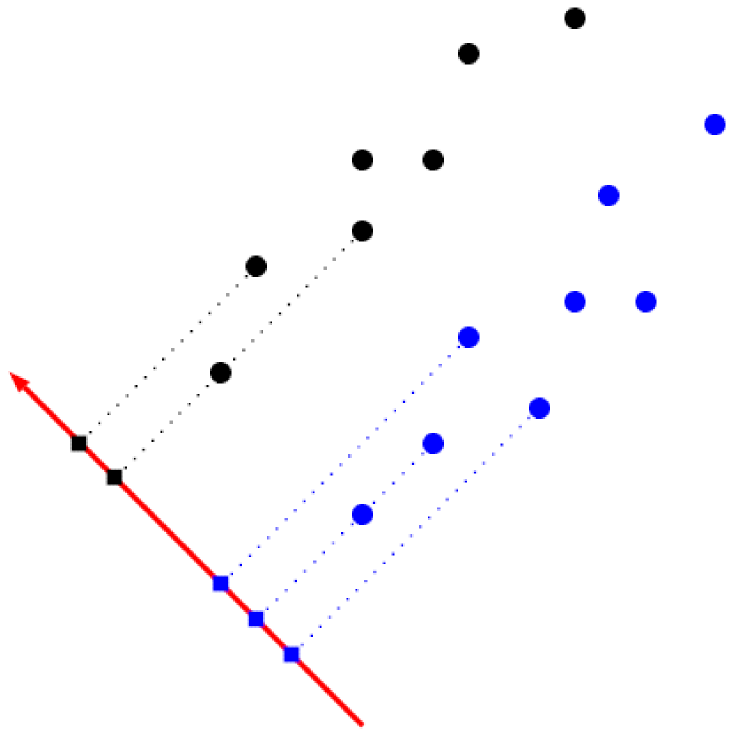


Representative but not discriminative

Linear Discriminant Analysis (LDA)

The two-class LDA problem

- Given a training data set $x_1, \ldots, x_n \in R^d$ consisting of two classes C_1, C_2 find a (unit-vector) direction that "best" discriminates between the two classes.

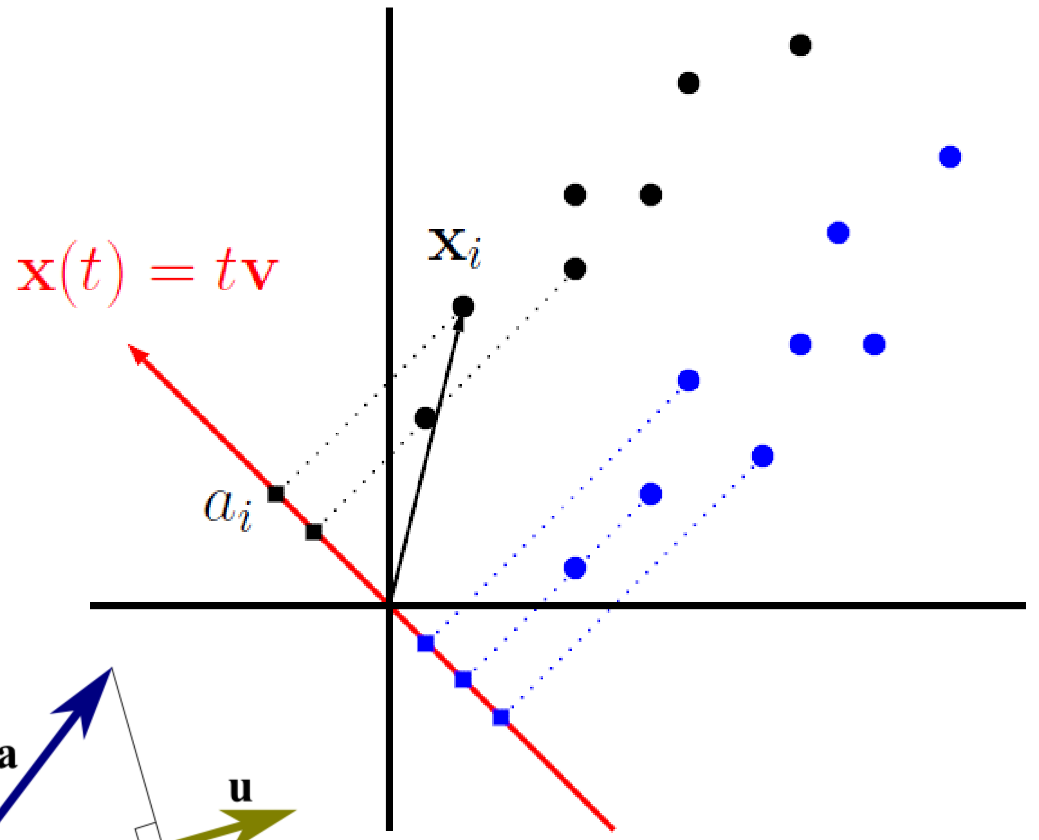


The goal of LDA is to reduce the dimensionality to 1–D where it finds a linear projection where the projected observations are well separated.

Linear Discriminant Analysis (LDA)

This time we are going to focus on lines that pass through the origin.

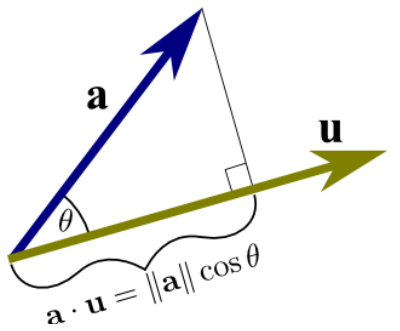
Consider any unit vector v ∈ R^d:



The 1D projections of the points are

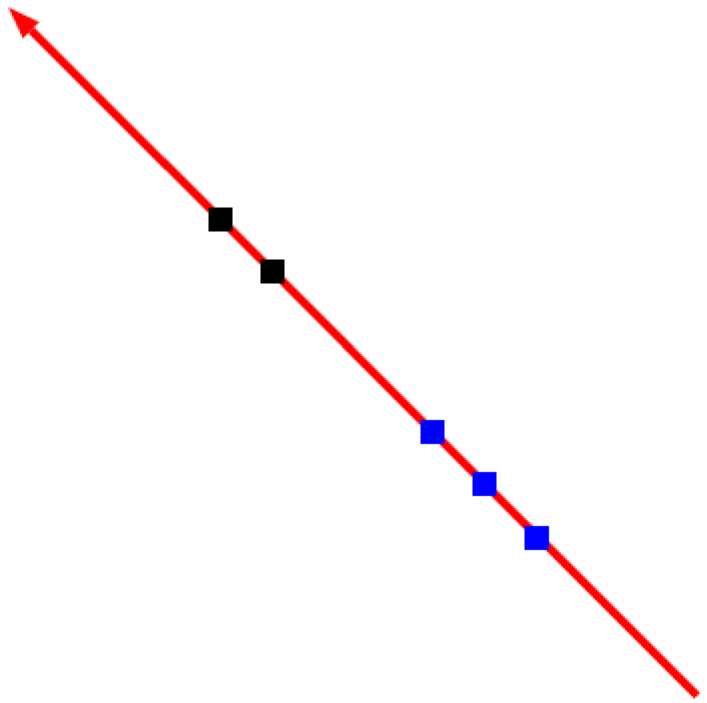
$$a_i = \mathbf{v}^T \mathbf{x}_i, \quad i = 1, \ldots, n$$

Note that they also carry the labels of the original data.



Linear Discriminant Analysis (LDA)

Now the data look like this:



How do we quantify the separation between the two classes (in order to compare different directions v and select the best one)?

One (naive) idea is to measure the distance between the two class means in the 1D projection space: $|\mu_1 - \mu_2|$, where

$$\mu_1 = \frac{1}{n_1} \sum_{\mathbf{x}_i \in C_1} a_i = \frac{1}{n_1} \sum_{\mathbf{x}_i \in C_1} \mathbf{v}^T \mathbf{x}_i$$
$$= \mathbf{v}^T \cdot \frac{1}{n_1} \sum_{\mathbf{x}_i \in C_1} \mathbf{x}_i = \mathbf{v}^T \mathbf{m}_1$$

and similarly,

$$\mu_2 = \mathbf{v}^T \mathbf{m}_2, \quad \mathbf{m}_2 = \frac{1}{n_2} \sum_{\mathbf{x}_i \in C_2} \mathbf{x}_i.$$

Linear Discriminant Analysis (LDA)

That is, we solve the following problem

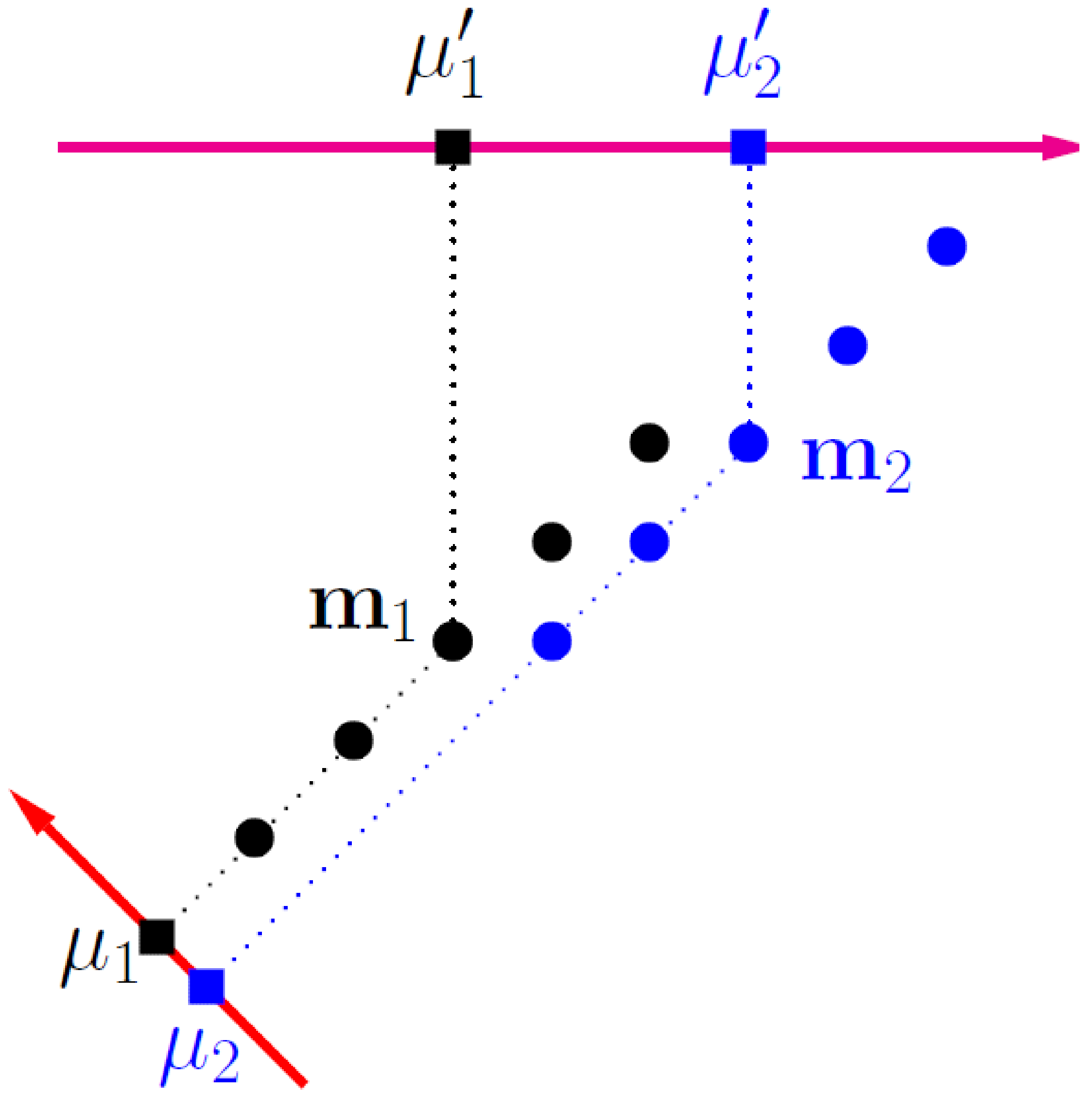
$$\max_{\mathbf{v}:\,\|\mathbf{v}\|=1} |\mu_1 - \mu_2|$$

where

$$\mu_j = \mathbf{v}^T \mathbf{m}_j,\ \ j = 1, 2.$$

However, this criterion does not always work (as shown in the right plot).

What else do we need to control?



Linear Discriminant Analysis (LDA)

It turns out that we should also pay attention to the variances of the projected classes:

$$s_1^2 = \sum_{\mathbf{x}_i \in C_1} (a_i - \mu_1)^2, \quad s_2^2 = \sum_{\mathbf{x}_i \in C_2} (a_i - \mu_2)^2$$

Ideally, the projected classes have both faraway means and small variances.

This can be achieved through the following modified formulation:

$$\max_{\mathbf{v}:\|\mathbf{v}\|=1} \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}.$$

The optimal $\mathbf{v}$ should be such that

- $(\mu_1 - \mu_2)^2$: large
- s_1^2, s_2^2: both small

Linear Discriminant Analysis (LDA)

The optimal discriminatory direction is

$$\mathbf{v}^* = \mathbf{S}_w^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \quad \text{(plus normalization)}$$

where

$$\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2, \quad \mathbf{S}_j = \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T$$

Linear Discriminant Analysis (LDA)

A small example

Data

- Class 1 has three points (1,2), (2,3), (3, 4.9), with mean $\mathbf{m}_1 = (2, 3.3)^T$
- Class 2 has three points (2,1), (3,2), (4, 3.9), with mean $\mathbf{m}_2 = (3, 2.3)^T$

Within-class scatter matrix

$$\mathbf{S}_w = \begin{pmatrix} 4 & 5.8 \\ 5.8 & 8.68 \end{pmatrix}$$

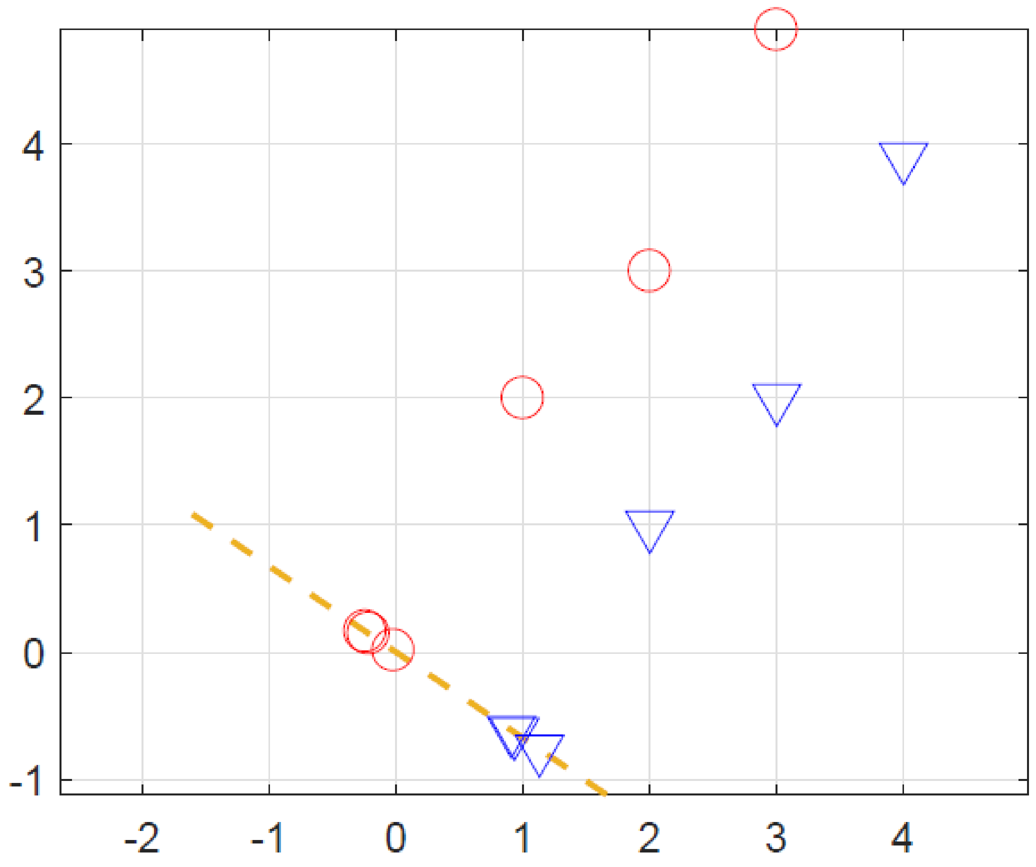
Thus, the optimal direction is

$$\mathbf{v} = \mathbf{S}_w^{-1}(\mathbf{m}_1 - \mathbf{m}_2) = (-13.4074, 9.0741)^T \xrightarrow{\text{normalizing}} (-0.8282, 0.5605)^T$$

Linear Discriminant Analysis (LDA)

and the projection coordinates are

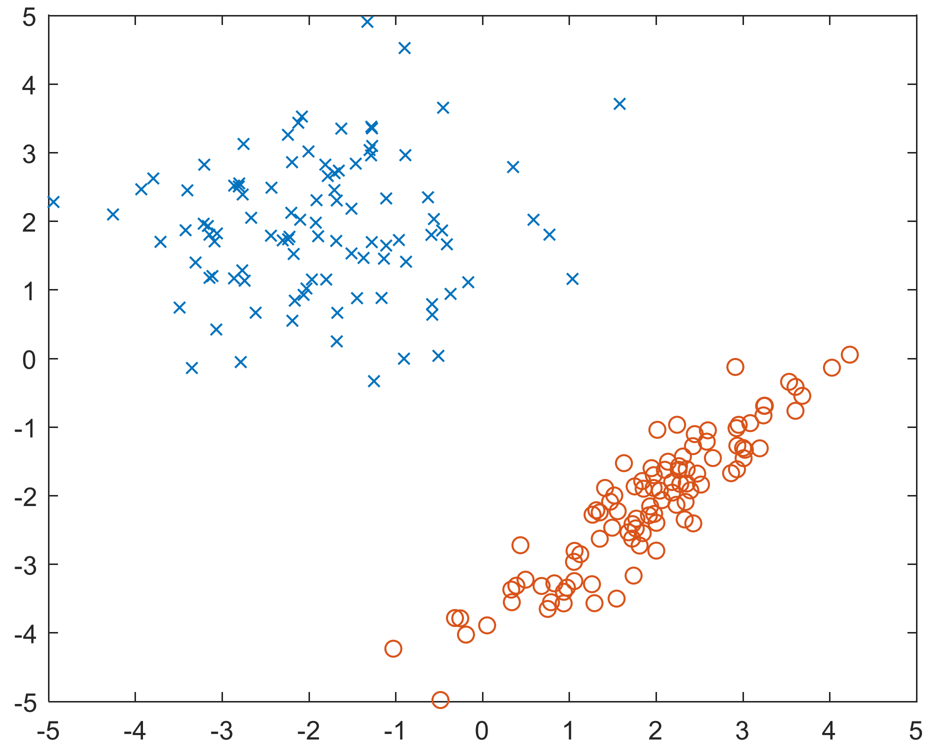
$$Y = [0.2928, 0.0252, 0.2619, -1.0958, -1.3635, -1.1267]$$



Linear Discriminant Analysis (LDA)

```
n1 = 100;
n2 = 100;
cov1 = eye(2);
cov2 = [1 .9; .9 1];
% The mvnrnd function is in the Statistics
Toolbox.
dat1 = mvnrnd([-2 2],cov1,n1);
dat2 = mvnrnd([2 -2],cov2,n2);
plot(dat1(:,1),dat1(:,2),'x',...
dat2(:,1),dat2(:,2),'o')


% Calculate the scatter matrices, using
the fact that
% they are proportional
% to the sample covariance matrices.
scat1 = (n1-1)*cov(dat1);
scat2 = (n2-1)*cov(dat2);
% Now we compute the within-class scatter
matrix.
Sw = scat1 + scat2;
```



Linear Discriminant Analysis (LDA)

```
% Next, we need the means for each class in the
% original space.
mu1 = mean(dat1);
mu2 = mean(dat2);


% The next steps calculate the LDA vector v.
% Note that we need to take the transpose of
% the means, since they need to be column vectors.
v = inv(Sw)*(mu1' - mu2');
% Normalize the vector v.
v = v/norm(v);


% We can now calculate the projected data.
pdat1 = v'*dat1';
pdat2 = v'*dat2';
```

Thank You

&

Questions