Algorithm Foundations of Data Science and Engineering

Lecture 5: SVD and PCA

MING GAO

DaSE @ ECNU
(for course related communications)
mgao@dase.ecnu.edu.cn

Mar. 25, 2019

Outline

High-dimensional classifier

We would like to create a classifier that is able to distinguish dogs from cats automatically from a set of images.

High-dimensional classifier

We would like to create a classifier that is able to distinguish dogs from cats automatically from a set of images.

- A possible descriptor that discriminates these two classes could then consist of three number: the average red color, the average green color, and the average blue color of the image.

High-dimensional classifier

We would like to create a classifier that is able to distinguish dogs from cats automatically from a set of images.

- A possible descriptor that discriminates these two classes could then consist of three number: the average red color, the average green color, and the average blue color of the image.
- A simple linear classifier for instance, could combine these features linearly to decide on the class label:

$$\text{Object is } \begin{cases} cat, & \text{if } 0.5 \times red + 0.3 \times green + 0.2 \times blue > 0.6; \\ dog, & \text{otherwise.} \end{cases}$$

High-dimensional classifier

We would like to create a classifier that is able to distinguish dogs from cats automatically from a set of images.

- A possible descriptor that discriminates these two classes could then consist of three number: the average red color, the average green color, and the average blue color of the image.
- A simple linear classifier for instance, could combine these features linearly to decide on the class label:

$$\text{Object is} \begin{cases} \mathit{cat}, & \text{if } 0.5 \times \mathit{red} + 0.3 \times \mathit{green} + 0.2 \times \mathit{blue} > 0.6; \\ \mathit{dog}, & \text{otherwise.} \end{cases}$$

- However, these three color-describing numbers, called features, will obviously not suffice to obtain a perfect classification.

High-dimensional classifier

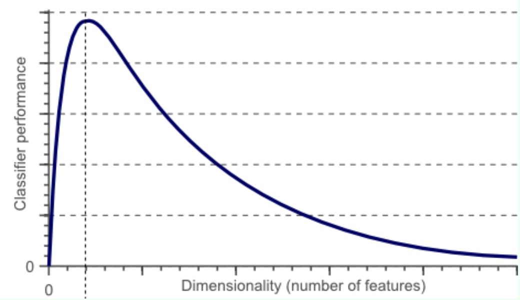
We would like to create a classifier that is able to distinguish dogs from cats automatically from a set of images.

- A possible descriptor that discriminates these two classes could then consist of three number: the average red color, the average green color, and the average blue color of the image.
- A simple linear classifier for instance, could combine these features linearly to decide on the class label:

$$\text{Object is } \begin{cases} cat, & \text{if } 0.5 \times red + 0.3 \times green + 0.2 \times blue > 0.6; \\ dog, & \text{otherwise.} \end{cases}$$

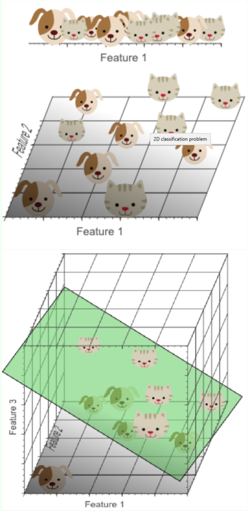
- However, these three color-describing numbers, called features, will obviously not suffice to obtain a perfect classification.
- Maybe we can obtain a perfect classification by carefully defining a few hundred of these features?

Classifier performance

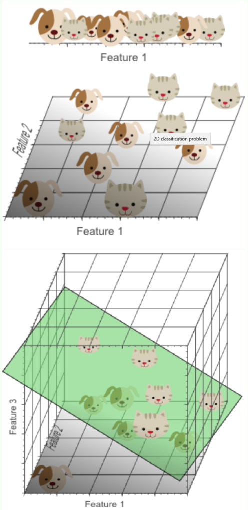


As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached. Further increasing the dimensionality without increasing the number of training samples results in a decrease in classifier performance.

Adding features

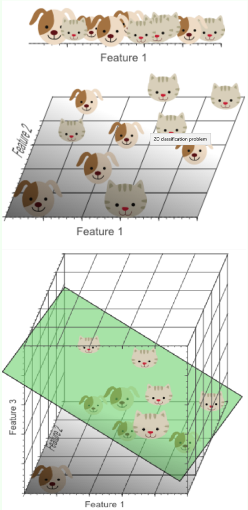


Adding features



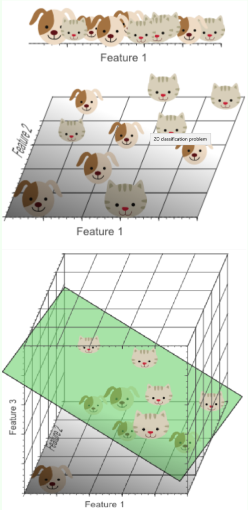
- A single feature does not result in a perfect separation of our training data.

Adding features



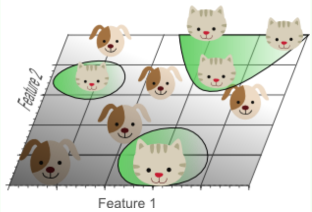
- A single feature does not result in a perfect separation of our training data.
- Adding a second feature still does not result in a linearly separable classification problem: No single line can separate all cats from all dogs in this example.

Adding features



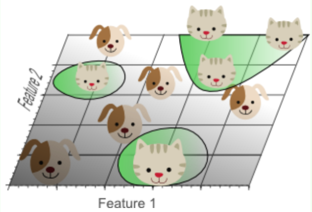
- A single feature does not result in a perfect separation of our training data.
- Adding a second feature still does not result in a linearly separable classification problem: No single line can separate all cats from all dogs in this example.
- The more features we use, the higher the likelihood that we can successfully separate the classes perfectly.

Non-separable classification in low-dimensional space



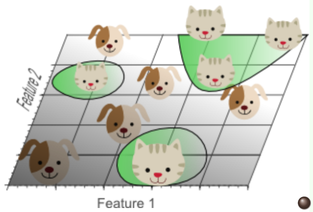
- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.

Non-separable classification in low-dimensional space



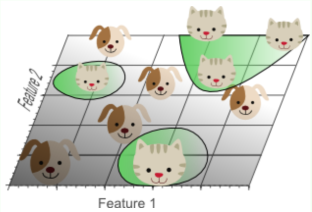
- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.
- Due to this sparsity, it becomes much more easy to find a separable hyperplane because the likelihood that a training sample lies on the wrong side of the best hyperplane becomes infinitely small when the number of features becomes infinitely large.

Non-separable classification in low-dimensional space

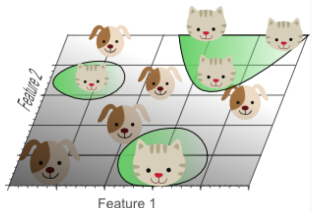


- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.
- Due to this sparsity, it becomes much more easy to find a separable hyperplane because the likelihood that a training sample lies on the wrong side of the best hyperplane becomes infinitely small when the number of features becomes infinitely large.
- Figure 6 shows the 3D classification results, projected onto a 2D feature space. Whereas the data was linearly separable in the 3D space, this is not the case in a lower dimensional feature space.

Non-separable classification in low-dimensional space

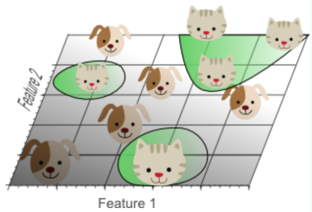


Non-separable classification in low-dimensional space



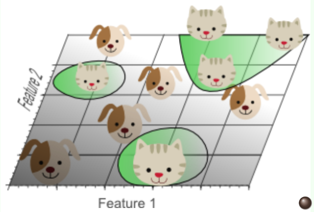
- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.

Non-separable classification in low-dimensional space



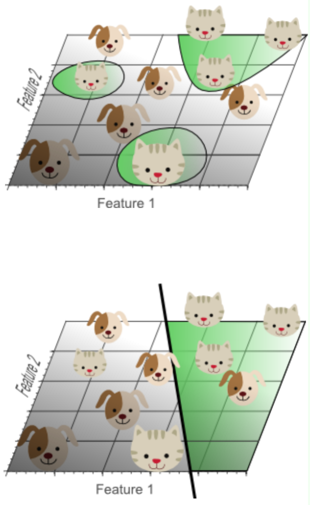
- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.
- Due to this sparsity, it becomes much more easy to find a separable hyperplane because the likelihood that a training sample lies on the wrong side of the best hyperplane becomes infinitely small when the number of features becomes infinitely large.

Non-separable classification in low-dimensional space

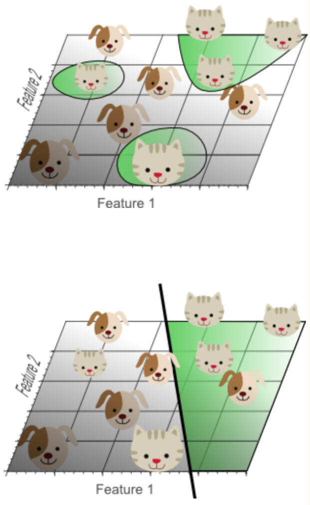


- If we would keep adding features, the dimensionality of the feature space grows, and becomes sparser and sparser.
- Due to this sparsity, it becomes much more easy to find a separable hyperplane because the likelihood that a training sample lies on the wrong side of the best hyperplane becomes infinitely small when the number of features becomes infinitely large.
- Figure 6 shows the 3D classification results, projected onto a 2D feature space. Whereas the data was linearly separable in the 3D space, this is not the case in a lower dimensional feature space.

Overfitting

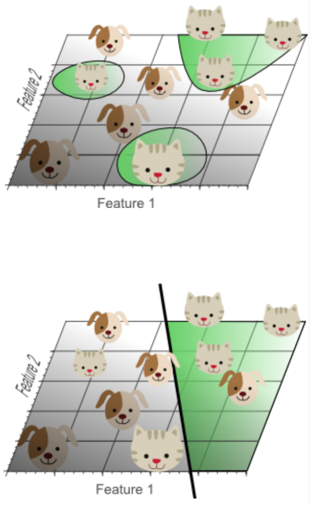


Overfitting



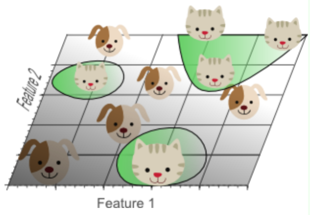
- In fact, adding the third dimension to obtain perfect classification results, simply corresponds to using a complicated non-linear classifier in the lower dimensional feature space.

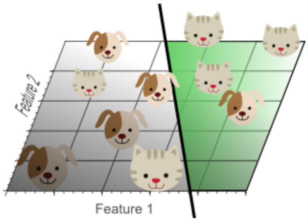
Overfitting



- In fact, adding the third dimension to obtain perfect classification results, simply corresponds to using a complicated non-linear classifier in the lower dimensional feature space.
- Because of this, the resulting classifier would fail on real-world data, consisting of an infinite amount of unseen cats and dogs that often do not adhere to these exceptions.

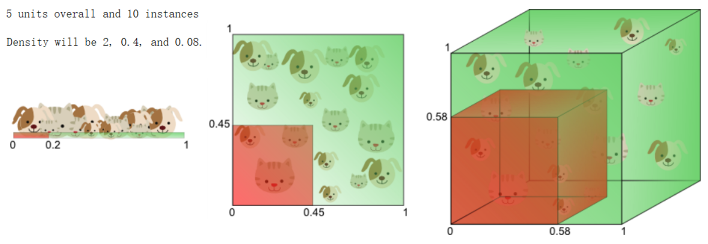
Overfitting



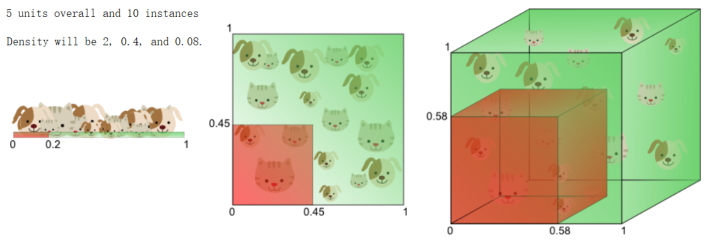


- In fact, adding the third dimension to obtain perfect classification results, simply corresponds to using a complicated non-linear classifier in the lower dimensional feature space.
- Because of this, the resulting classifier would fail on real-world data, consisting of an infinite amount of unseen cats and dogs that often do not adhere to these exceptions.
- This concept is called overfitting and is a direct result of the curse of dimensionality.

Overfitting

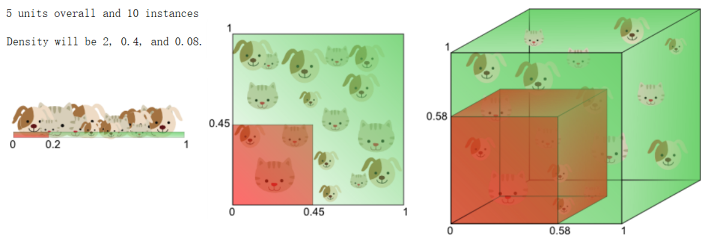


Overfitting



- For 2D feature space, to cover 20% of the 2D feature range, we now need to obtain 45% of the complete population ($0.45^2 = 0.2$). In the 3D case this gets even worse, the proportion will be 58% ($0.58^3 = 0.2$).

Overfitting

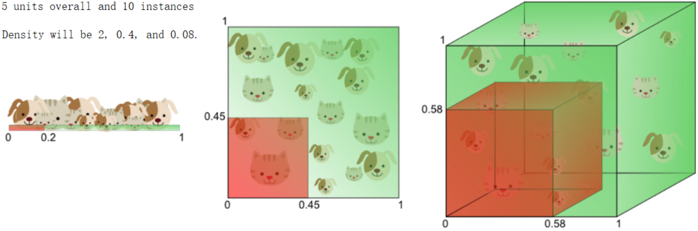


- For 2D feature space, to cover 20% of the 2D feature range, we now need to obtain 45% of the complete population ($0.45^2 = 0.2$). In the 3D case this gets even worse, the proportion will be 58% ($0.58^3 = 0.2$).
- If we keep adding dimensions, the amount of training data needs to grow exponentially fast to maintain the same coverage and to avoid overfitting.

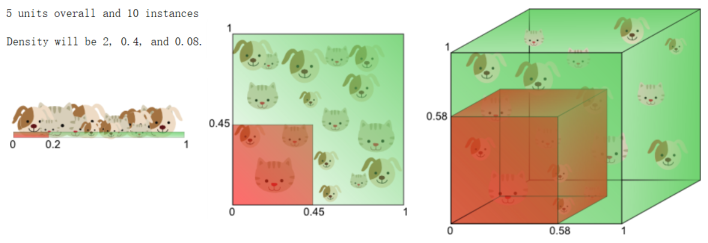
Overfitting

5 units overall and 10 instances

Density will be 2, 0.4, and 0.08.

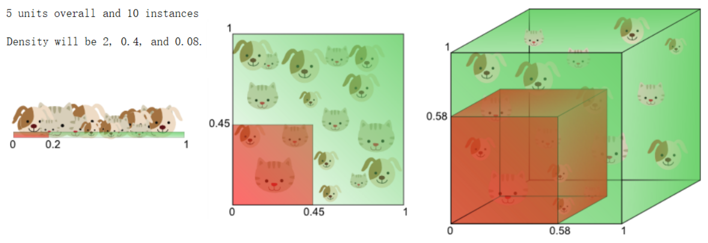


Overfitting



- For 2D feature space, to cover 20% of the 2D feature range, we now need to obtain 45% of the complete population ($0.45^2 = 0.2$). In the 3D case this gets even worse, the proportion will be 58% ($0.58^3 = 0.2$).

Overfitting



- For 2D feature space, to cover 20% of the 2D feature range, we now need to obtain 45% of the complete population ($0.45^2 = 0.2$). In the 3D case this gets even worse, the proportion will be 58% ($0.58^3 = 0.2$).
- If we keep adding dimensions, the amount of training data needs to grow exponentially fast to maintain the same coverage and to avoid overfitting.

How to avoid the curse of dimensionality

- The performance of a classifier decreases when the dimensionality becomes too large. The questions then are what "too large" means, and how overfitting can be avoided.

How to avoid the curse of dimensionality

- The performance of a classifier decreases when the dimensionality becomes too large. The questions then are what "too large" means, and how overfitting can be avoided.
- Regrettably there is no fixed rule that defines how many feature should be used in a classification problem.

How to avoid the curse of dimensionality

- The performance of a classifier decreases when the dimensionality becomes too large. The questions then are what "too large" means, and how overfitting can be avoided.
- Regrettably there is no fixed rule that defines how many feature should be used in a classification problem.
- In fact, this depends on the amount of training data available, the complexity of the decision boundaries, and the type of classifier used.

How to avoid the curse of dimensionality

- The performance of a classifier decreases when the dimensionality becomes too large. The questions then are what "too large" means, and how overfitting can be avoided.
- Regrettably there is no fixed rule that defines how many feature should be used in a classification problem.
- In fact, this depends on the amount of training data available, the complexity of the decision boundaries, and the type of classifier used.
- **feature selection** would be to search for the optimal set of features. The method often employs heuristics (greedy methods, best-first methods, etc.) to locate the optimal number and combination of features.

How to avoid the curse of dimensionality

- The performance of a classifier decreases when the dimensionality becomes too large. The questions then are what "too large" means, and how overfitting can be avoided.
- Regrettably there is no fixed rule that defines how many feature should be used in a classification problem.
- In fact, this depends on the amount of training data available, the complexity of the decision boundaries, and the type of classifier used.
- **feature selection** would be to search for the optimal set of features. The method often employs heuristics (greedy methods, best-first methods, etc.) to locate the optimal number and combination of features.
- The other approach, **Dimensionality reduction**, would be to replace the set of N features by a set of M features, each of which is a combination of the original feature values.

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

- Netflix: 480K users and 177K movies

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

- Netflix: 480K users and 177K movies
- Documents VS. words: thousands of words and billions of documents

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

- Netflix: 480K users and 177K movies
- Documents VS. words: thousands of words and billions of documents
- Product adoption: millions of users and millions of products

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

- Netflix: 480K users and 177K movies
- Documents VS. words: thousands of words and billions of documents
- Product adoption: millions of users and millions of products

Huge number of dimensions causes problems

- Data becomes very sparse, some algorithms become meaningless (e.g. density based clustering)

The curse of dimensionality

Real data usually have thousands, or millions of dimensions.

- Netflix: 480K users and 177K movies
- Documents VS. words: thousands of words and billions of documents
- Product adoption: millions of users and millions of products

Huge number of dimensions causes problems

- Data becomes very sparse, some algorithms become meaningless (e.g. density based clustering)
- The complexity of several algorithms depends on the dimensionality and they become infeasible.

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data
- Approaches for dimensionality reduction

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data
- Approaches for dimensionality reduction
  - LSH

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data
- Approaches for dimensionality reduction
  - LSH
  - Topic modeling

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data
- Approaches for dimensionality reduction
  - LSH
  - Topic modeling
  - Neural embedding

Dimensionality reduction

Usually the data can be described with fewer dimensions, without losing much of the meaning of the data.

- The data reside in a space of lower dimensionality
- Assume that some of the data is noise, and we can approximate the useful part with a lower dimensionality space.
- Dimensionality reduction does not just reduce the amount of data, it often brings out the useful part of the data.
  - Discover latent relations between objects.
  - Remove redundant and noisy features, not all features are useful
  - Easier storage and processing of the data
- Approaches for dimensionality reduction
  - LSH
  - Topic modeling
  - Neural embedding
  - Matrix decomposition, such as SVD, PCA, MF, PMF, etc

Eigenvalue and eigenvector

Definition

Given a square matrix $A \in \mathbb{R}^{n \times n}$, and non-zero column vector $\mathbf{v}$, if

$$A\mathbf{v} = \lambda\mathbf{v},$$

where λ is an eigenvalue of A and $\mathbf{v}$ is an eigenvector corresponding to eigenvalue λ.

Eigenvalue and eigenvector

Definition

Given a square matrix $A \in \mathbb{R}^{n \times n}$, and non-zero column vector $\mathbf{v}$, if

$$A\mathbf{v} = \lambda \mathbf{v},$$

where λ is an eigenvalue of A and $\mathbf{v}$ is an eigenvector corresponding to eigenvalue λ.

- For example,

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = 4 \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Eigenvalue and eigenvector

Definition

Given a square matrix $A \in \mathbb{R}^{n \times n}$, and non-zero column vector $\mathbf{v}$, if

$$A\mathbf{v} = \lambda \mathbf{v},$$

where λ is an eigenvalue of A and $\mathbf{v}$ is an eigenvector corresponding to eigenvalue λ.

- For example,

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = 4 \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

- If we think of the squared matrix A as a transformation matrix, then multiply it with the eigenvector do not change its direction.

Outline

Diagonalization

Definition of similar matrices

If matrices $A, B \in \mathbb{R}^{n \times n}$ are said to be similar if there is an invertible $n \times n$ matrix P such that $A = PBP^{-1}$.

Diagonalization

Definition of similar matrices

If matrices $A, B \in \mathbb{R}^{n \times n}$ are said to be similar if there is an invertible $n \times n$ matrix P such that $A = PBP^{-1}$.

- Theorem: If $n \times n$ matrices are similar, then they have the same characteristic of polynomial and hence the same enigenvalues (with the same multiplicities).

Diagonalization

Definition of similar matrices

If matrices $A, B \in \mathbb{R}^{n\times n}$ are said to be similar if there is an invertible $n \times n$ matrix P such that $A = PBP^{-1}$.

- Theorem: If $n \times n$ matrices are similar, then they have the same characteristic of polynomial and hence the same enigenvalues (with the same multiplicities).
- A square matrix A is said to be diagonalizable if it is similar to a diagonal matrix, i.e., there exists an invertible matrix P and a diagonal matrix D such that $A = PDP^{-1}$.

Diagonalization

Definition of similar matrices

If matrices $A, B \in \mathbb{R}^{n \times n}$ are said to be similar if there is an invertible $n \times n$ matrix P such that $A = PBP^{-1}$.

- Theorem: If $n \times n$ matrices are similar, then they have the same characteristic of polynomial and hence the same enigenvalues (with the same multiplicities).
- A square matrix A is said to be diagonalizable if it is similar to a diagonal matrix, i.e., there exists an invertible matrix P and a diagonal matrix D such that $A = PDP^{-1}$.
    - Why useful? If A is diagonalizable, then $A^k = PD^kP^{-1}$ for $k > 0$.

Diagonalization

Definition of similar matrices

If matrices $A, B \in \mathbb{R}^{n\times n}$ are said to be similar if there is an invertible $n \times n$ matrix P such that $A = PBP^{-1}$.

- Theorem: If $n \times n$ matrices are similar, then they have the same characteristic of polynomial and hence the same enigenvalues (with the same multiplicities).
- A square matrix A is said to be diagonalizable if it is similar to a diagonal matrix, i.e., there exists an invertible matrix P and a diagonal matrix D such that $A = PDP^{-1}$.
  - Why useful? If A is diagonalizable, then $A^k = PD^kP^{-1}$ for $k > 0$.
  - How to find diagonal matrix? If $v_1, \cdots, v_n$ are linearly independent eigenvectors of A and λ_i are their corresponding eignevalues, then $A = PDP^{-1}$, where $P = [v_1 \ \cdots \ v_n]$ and $D = Diag(\lambda_1, \cdots, \lambda_n)$.

Conditions of diagonalizable

Theorem

- A matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable if and only if A has n linearly independent eigenvectors.

Conditions of diagonalizable

Theorem

- A matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable if and only if A has n linearly independent eigenvectors.
- A matrix $A \in \mathbb{R}^{n \times n}$ with n distinct eigenvalues is diagonalizable.

Conditions of diagonalizable

Theorem

- A matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable if and only if A has n linearly independent eigenvectors.
- A matrix $A \in \mathbb{R}^{n \times n}$ with n distinct eigenvalues is diagonalizable.

Example:

- $A = \begin{bmatrix} 2 & 4 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{bmatrix}$ with $\lambda_1 = 2$ and $\lambda_1 = 4$.

Conditions of diagonalizable

Theorem

- A matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable if and only if A has n linearly independent eigenvectors.
- A matrix $A \in \mathbb{R}^{n \times n}$ with n distinct eigenvalues is diagonalizable.

Example:

- $A = \begin{bmatrix} 2 & 4 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{bmatrix}$ with $\lambda_1 = 2$ and $\lambda_1 = 4$.
- While the basis for $\lambda_1 = 2$ is $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $((A - 2I)\mathbf{v} = 0)$.

Conditions of diagonalizable

Theorem

- A matrix $A \in \mathbb{R}^{n\times n}$ is diagonalizable if and only if A has n linearly independent eigenvectors.
- A matrix $A \in \mathbb{R}^{n\times n}$ with n distinct eigenvalues is diagonalizable.

Example:

- $A = \begin{bmatrix} 2 & 4 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{bmatrix}$ with $\lambda_1 = 2$ and $\lambda_1 = 4$.
- While the basis for $\lambda_1 = 2$ is $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $((A - 2I)\mathbf{v} = 0)$.
- Hence, the matrix is not diagonalizable.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Properties of symmetric matrix:

- If A is symmetric then any two eigenvalues from different eigenspaces are orthogonal.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Properties of symmetric matrix:

- If A is symmetric then any two eigenvalues from different eigenspaces are orthogonal.
- A has n real eigenvalues if we count multiplicity.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Properties of symmetric matrix:

- If A is symmetric then any two eigenvalues from different eigenspaces are orthogonal.
- A has n real eigenvalues if we count multiplicity.
- For each eigenvalue, the dimension of the corresponding eigenspace is equal to the algebraic multiplicity of that eigenvalue.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Properties of symmetric matrix:

- If A is symmetric then any two eigenvalues from different eigenspaces are orthogonal.
- A has n real eigenvalues if we count multiplicity.
- For each eigenvalue, the dimension of the corresponding eigenspace is equal to the algebraic multiplicity of that eigenvalue.
- The eigenspaces are mutually orthogonal.

Diagonalization of of symmetric matrix

Definition of orthogonal

- Matrix P is orthogonal if $P^{-1} = P^T$.
- Matrix A is orthogonal diagonalizable if there is a square matrix P such that $A = PDP^T$ where D is a diagonal matrix.

Properties of symmetric matrix:

- If A is symmetric then any two eigenvalues from different eigenspaces are orthogonal.
- A has n real eigenvalues if we count multiplicity.
- For each eigenvalue, the dimension of the corresponding eigenspace is equal to the algebraic multiplicity of that eigenvalue.
- The eigenspaces are mutually orthogonal.
- A is orthogonal diagonalizable.

Geometric explanation of diagonalization

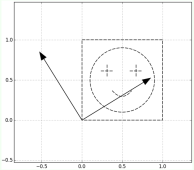
Example

$$\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix} \begin{bmatrix} 1.81 & 0 \\ 0 & 0.69 \end{bmatrix} \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix}^T$$

Geometric explanation of diagonalization

Example

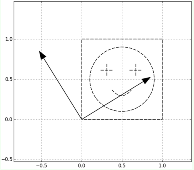
$$\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix} \begin{bmatrix} 1.81 & 0 \\ 0 & 0.69 \end{bmatrix} \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix}^T$$

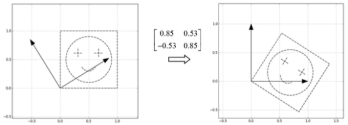


Geometric explanation of diagonalization

Example

$$\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix} \begin{bmatrix} 1.81 & 0 \\ 0 & 0.69 \end{bmatrix} \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix}^T$$

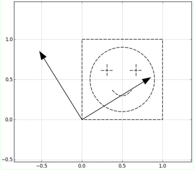


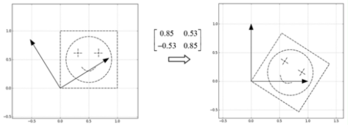


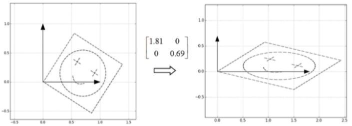
Geometric explanation of diagonalization

Example

$$\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix} \begin{bmatrix} 1.81 & 0 \\ 0 & 0.69 \end{bmatrix} \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix}^T$$



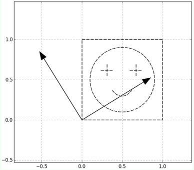


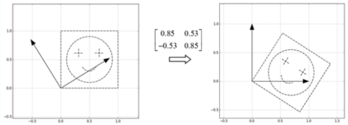


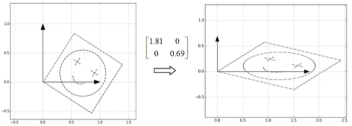
Geometric explanation of diagonalization

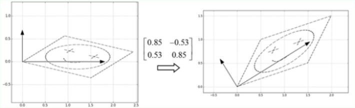
Example

$$\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix} = \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix} \begin{bmatrix} 1.81 & 0 \\ 0 & 0.69 \end{bmatrix} \begin{bmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{bmatrix}^T$$









Outline

1. The Curse of Dimensionality
2. Singular Value Decomposition
   - Diagonalization
   - Singular Value Decomposition
3. Principal Component Analysis

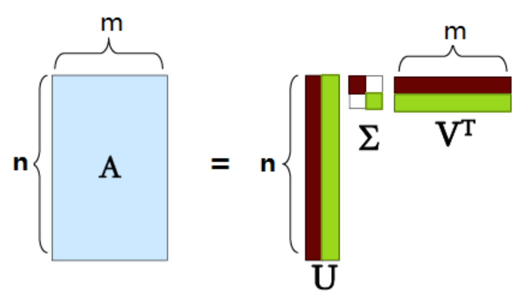
Singular value decomposition: SVD

Any real $m \times n$ matrix A can be decomposed uniquely as

Singular value decomposition: SVD

Any real $m \times n$ matrix A can be decomposed uniquely as

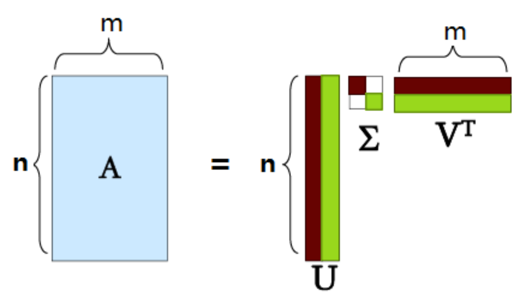
$$A_{[n \times m]} \sim U_{[n \times r]} D_{[r \times r]} V^T_{r \times m}$$



Singular value decomposition: SVD

Any real $m \times n$ matrix A can be decomposed uniquely as

$$A_{[n \times m]} \sim U_{[n \times r]} D_{[r \times r]} V^T_{r \times m}$$

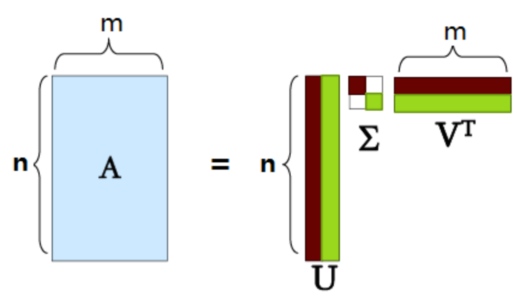


- A: an input $n \times m$ data matrix (may not be a square matrix), e.g., n users and m items.

Singular value decomposition: SVD

Any real $m \times n$ matrix A can be decomposed uniquely as

$$A_{[n \times m]} \sim U_{[n \times r]} D_{[r \times r]} V^T_{r \times m}$$

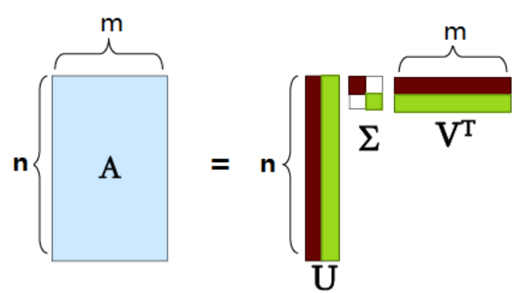


- A: an input $n \times m$ data matrix (may not be a square matrix), e.g., n users and m items.
- U and V: left (n users and r interests) and right singular (r interests and m items) vectors.

Singular value decomposition: SVD

Any real $m \times n$ matrix A can be decomposed uniquely as

$$A_{[n\times m]} \sim U_{[n\times r]} D_{[r\times r]} V^T_{r\times m}$$



- A: an input $n \times m$ data matrix (may not be a square matrix), e.g., n users and m items.
- U and V: left (n users and r interests) and right singular (r interests and m items) vectors.
- D: a $r \times r$ diagonal matrix, e.g., strength of each interest.

SVD Cont'd

It is always possible to decompose a real matrix A into $A = UDV^T$

SVD Cont'd

It is always possible to decompose a real matrix A into $A = UDV^T$

- U, D, V are unique, and D is a diagonal matrix.

SVD Cont'd

It is always possible to decompose a real matrix A into $A = UDV^T$

- U, D, V are unique, and D is a diagonal matrix.
- U, V are column orthogonal (i.e., $U^T U = I$ and $V^T V = I$)

SVD Cont'd

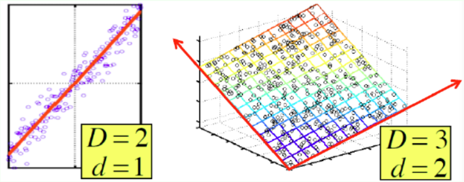
It is always possible to decompose a real matrix A into $A = UDV^T$

- U, D, V are unique, and D is a diagonal matrix.
- U, V are column orthogonal (i.e., $U^T U = I$ and $V^T V = I$)
- Entries of D are positive and sorted in decreasing order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$, where k is rank of matrix A.

SVD Cont'd

It is always possible to decompose a real matrix A into $A = UDV^T$

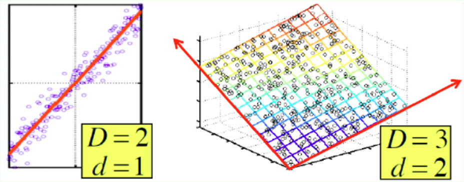
- U, D, V are unique, and D is a diagonal matrix.
- U, V are column orthogonal (i.e., $U^T U = I$ and $V^T V = I$)
- Entries of D are positive and sorted in decreasing order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$, where k is rank of matrix A.



SVD Cont'd

It is always possible to decompose a real matrix A into $A = UDV^T$

- U, D, V are unique, and D is a diagonal matrix.
- U, V are column orthogonal (i.e., $U^T U = I$ and $V^T V = I$)
- Entries of D are positive and sorted in decreasing order $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$, where k is rank of matrix A.



Assumptions

- Data lies on or near a low $d-$dimensional subspace.
- Axes of this subspace are effective representation of the data.

Methodology of decomposition

Diagonalization

- $$AA^T = UDV^TVDU^T = UD^2U^T,$$

  where $D = diag(\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2)$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$.

Methodology of decomposition

Diagonalization

- $$AA^T = UDV^TVDU^T = UD^2U^T,$$
- where $D = diag(\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2)$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$.

  $$A^TA = VDU^TUDV^T = VD^2V^T, \text{ i.e., } V = A^TUD^{-1}.$$

Methodology of decomposition

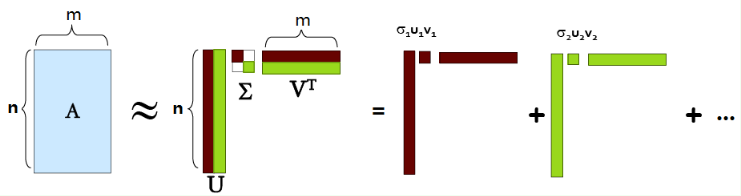
Diagonalization

- $$AA^T = UDV^TVDU^T = UD^2U^T,$$
- where $D = diag(\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2)$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$.
  $$A^TA = VDU^TUDV^T = VD^2V^T, \text{ i.e., } V = A^TUD^{-1}.$$
- $A \approx A^{(k)} = UDV^T$, if $U := (\mathbf{u}_1\ \mathbf{u}_2\ \cdots \mathbf{u}_k)$ and $V := (\mathbf{v}_1\ \mathbf{v}_2\ \cdots \mathbf{v}_k)$, then $A^{(k)} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T$.

Methodology of decomposition

Diagonalization

- $$AA^T = UDV^TVDU^T = UD^2U^T,$$
- where $D = diag(\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2)$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$.
  $$A^TA = VDU^TUDV^T = VD^2V^T, \text{ i.e., } V = A^TUD^{-1}.$$
- $A \approx A^{(k)} = UDV^T$, if $U := (\mathbf{u}_1\ \mathbf{u}_2\ \cdots \mathbf{u}_k)$ and $V := (\mathbf{v}_1\ \mathbf{v}_2\ \cdots \mathbf{v}_k)$, then $A^{(k)} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T$.



Methodology of dimensionality reduction

$$A = \left[\begin{array}{ccc|ccc} u_1 & \cdots & u_k & u_{k+1} & \cdots & u_m \end{array}\right] \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_k & \\ \hline & 0 & & 0 \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \\ \hline v_{k+1}^T \\ \vdots \\ v_n^T \end{array}\right]$$

$$A = \left[\begin{array}{ccc} u_1 & \cdots & u_k \end{array}\right] \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \end{array}\right]$$

Methodology of dimensionality reduction

$$A = \left[\begin{array}{ccc|ccc} u_1 & \cdots & u_k & u_{k+1} & \cdots & u_m \end{array}\right] \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_k & \\ \hline & 0 & & 0 \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \\ \hline v_{k+1}^T \\ \vdots \\ v_n^T \end{array}\right]$$

$$A = \left[\begin{array}{ccc} u_1 & \cdots & u_k \end{array}\right] \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \end{array}\right]$$

- $r = \arg\min_l \left\{ \frac{\sum_{i=1}^{l} \sigma_i}{\sum_{i=1}^{k} \sigma_i} > 90\% \right\},\ A^{(r)} \approx \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$

Methodology of dimensionality reduction

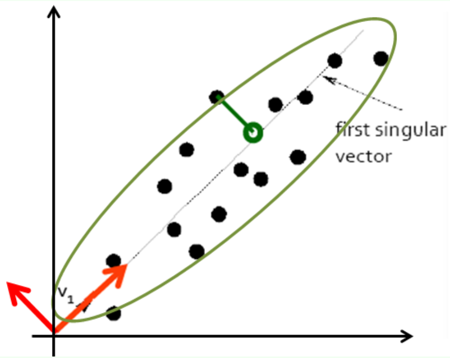
$$A = \left[\begin{array}{ccc|ccc} u_1 & \cdots & u_k & u_{k+1} & \cdots & u_m \end{array}\right] \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_k & \\ \hline & 0 & & 0 \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \\ \hline v_{k+1}^T \\ \vdots \\ v_n^T \end{array}\right]$$

$$A = \left[\begin{array}{ccc} u_1 & \cdots & u_k \end{array}\right] \left[\begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array}\right] \left[\begin{array}{c} v_1^T \\ \vdots \\ v_k^T \end{array}\right]$$

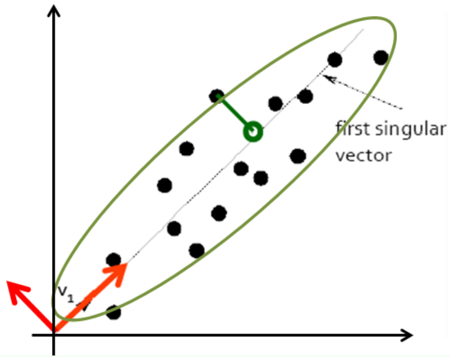
- $r = \arg\min_l \left\{ \frac{\sum_{i=1}^{l} \sigma_i}{\sum_{i=1}^{k} \sigma_i} > 90\% \right\}$, $A^{(r)} \approx \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$
- For example

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \approx \begin{bmatrix} 0.18 & 0 \\ 0.36 & 0 \\ 0.18 & 0 \\ 0.90 & 0 \\ 0 & 0.53 \\ 0 & 0.80 \\ 0 & 0.27 \end{bmatrix} \times \begin{bmatrix} 9.64 & 0 \\ 0 & 5.29 \end{bmatrix} \times \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix}$$

SVD interpretation

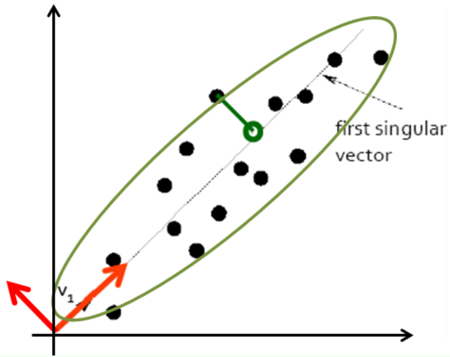


SVD interpretation



- SVD gives best axis to project entities, where "best" means to minimize sum of squares of projection errors.

SVD interpretation



- SVD gives best axis to project entities, where "best" means to minimize sum of squares of projection errors.
- SVD also gives the minimum reconstruction errors.

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

- $\|A\|_2 = \max_{x \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{x \neq 0} \sqrt{R_{A^T A}(\mathbf{x})}$;

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

- $\|A\|_2 = \max_{x \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{x \neq 0} \sqrt{R_{A^TA}(\mathbf{x})}$;
- $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2} = \sqrt{\mathrm{trace}(A^TA)}$;

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

- $\|A\|_2 = \max_{x \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{x \neq 0} \sqrt{R_{A^TA}(\mathbf{x})}$;
- $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2} = \sqrt{\text{trace}(A^TA)}$;

If $r < k$,

$$A^{(r)} \approx \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

The reconstruction error will be

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

- $\|A\|_2 = \max_{x \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{x \neq 0} \sqrt{R_{A^T A}(\mathbf{x})}$;
- $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2} = \sqrt{\text{trace}(A^T A)}$;

If $r < k$,

$$A^{(r)} \approx \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

The reconstruction error will be

$$\|A^{(r)} - A\|_2 = \|\sum_{i=r}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T\|_2 = \sigma_{r+1}.$$

Reconstruction error

Note that the 2-norm and the Frobenius norm of A are defined as

- $\|A\|_2 = \max_{x\neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{x\neq 0} \sqrt{R_{A^TA}(\mathbf{x})}$;
- $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} = \sqrt{\text{trace}(A^TA)}$;

If $r < k$,

$$A^{(r)} \approx \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

The reconstruction error will be

$$\|A^{(r)} - A\|_2 = \|\sum_{i=r}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T\|_2 = \sigma_{r+1}.$$

$$\|A^{(r)} - A\|_F = \|\sum_{i=r}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T\|_F = \sqrt{\sum_{i=r}^{k} \sigma_i^2}.$$

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)
  - If A is a nonsingular matrix, then its inverse is given by $A^{-1} = V^T D^{-1} U$.

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)
  - If A is a nonsingular matrix, then its inverse is given by $A^{-1} = V^T D^{-1} U$.
  - If A is singular or ill-conditioned, then we can use SVD to approximate its inverse by the following matrix: $A^{-1} = (UDV^T)^{-1} \approx VD_0^{-1}U^T$, where t is a small threshold and $D_0^{-1} = \begin{cases} \frac{1}{\sigma_i}, & \text{if } \sigma_i > t; \\ 0, & \text{otherwise.} \end{cases}$

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)
  - If A is a nonsingular matrix, then its inverse is given by $A^{-1} = V^T D^{-1} U$.
  - If A is singular or ill-conditioned, then we can use SVD to approximate its inverse by the following matrix: $A^{-1} = (UDV^T)^{-1} \approx VD_0^{-1}U^T$, where t is a small threshold and $D_0^{-1} = \begin{cases} \frac{1}{\sigma_i}, & \text{if } \sigma_i > t; \\ 0, & \text{otherwise.} \end{cases}$
- Consider linear system $Ax = b$, where $A \in \mathbb{R}^{n \times m}$. If $A^T A$ is ill-conditioned (small changes in b can lead to relatively large changes in the solution x) or singular, $x \approx VD_0^{-1}U^T b$.

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)
  - If A is a nonsingular matrix, then its inverse is given by $A^{-1} = V^T D^{-1} U$.
  - If A is singular or ill-conditioned, then we can use SVD to approximate its inverse by the following matrix: $A^{-1} = (UDV^T)^{-1} \approx VD_0^{-1}U^T$, where t is a small threshold and $D_0^{-1} = \begin{cases} \frac{1}{\sigma_i}, & \text{if } \sigma_i > t; \\ 0, & \text{otherwise.} \end{cases}$
- Consider linear system $Ax = b$, where $A \in \mathbb{R}^{n \times m}$. If $A^T A$ is ill-conditioned (small changes in b can lead to relatively large changes in the solution x) or singular, $x \approx VD_0^{-1}U^T b$.
- SVD can be helpful to similarity query or join.

Applications of SVD

Applications

- A square matrix A is nonsingular (i.e., $\sigma_i \neq 0$ for all i)
  - If A is a nonsingular matrix, then its inverse is given by $A^{-1} = V^T D^{-1} U$.
  - If A is singular or ill-conditioned, then we can use SVD to approximate its inverse by the following matrix: $A^{-1} = (UDV^T)^{-1} \approx VD_0^{-1}U^T$, where t is a small threshold and $D_0^{-1} = \begin{cases} \frac{1}{\sigma_i}, & \text{if } \sigma_i > t; \\ 0, & \text{otherwise.} \end{cases}$
- Consider linear system $Ax = b$, where $A \in \mathbb{R}^{n \times m}$. If $A^T A$ is ill-conditioned (small changes in b can lead to relatively large changes in the solution x) or singular, $x \approx VD_0^{-1}U^T b$.
- SVD can be helpful to similarity query or join.
- Data compression and anomaly detection.

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.
- Preserve as much information as possible

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.
- Preserve as much information as possible

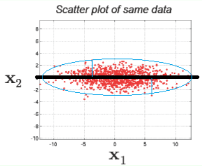
$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} --> reduce\ dimensionality --> y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.
- Preserve as much information as possible

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} --> reduce\ dimensionality --> y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$

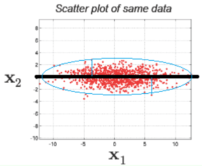


PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.
- Preserve as much information as possible

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} --> reduce\ dimensionality --> y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$



Goals

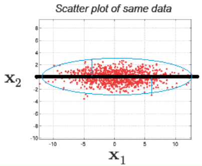
- Find a good representation for features (what?)

PCA: an important application of SVD

Motivation

- Problems arise in a high-dimensional space (e.g., curse of dimensionality).
- Significant improvements can be achieved by first mapping the data into a lower-dimensionality space.
- Preserve as much information as possible

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} -- > \textit{reduce dimensionality} -- > y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$



Goals

- Find a good representation for features (what?)
- Reduce redundancy in the data (how?)

PCA Cont'd

Criteria of good representation for features

- Minimize relation of the different dimensions.

PCA Cont'd

Criteria of good representation for features

- Minimize relation of the different dimensions.
- Keep the dimension as low as possible.

PCA Cont'd

Criteria of good representation for features

- Minimize relation of the different dimensions.
- Keep the dimension as low as possible.

The best low-dimensional space

- It also gives best axis to project data, where "best" means to minimize sum of squares of projection errors.

PCA Cont'd

Criteria of good representation for features

- Minimize relation of the different dimensions.
- Keep the dimension as low as possible.

The best low-dimensional space

- It also gives best axis to project data, where "best" means to minimize sum of squares of projection errors.
- It also gives the minimum reconstruction errors.

PCA Cont'd

Criteria of good representation for features

- Minimize relation of the different dimensions.
- Keep the dimension as low as possible.

The best low-dimensional space

- It also gives best axis to project data, where "best" means to minimize sum of squares of projection errors.
- It also gives the minimum reconstruction errors.
- It can be determined by the "best" eigenvectors of the covariance matrix of x (i.e., the eigenvectors corresponding to the "largest" eigenvalues, also called "principal components").

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

1. $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i$.

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

1. $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i$.
2. Subtract the mean: $\mathbf{y}_i = \mathbf{x}_i - \bar{\mathbf{x}}$.

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

1. $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i$.
2. Subtract the mean: $\mathbf{y}_i = \mathbf{x}_i - \bar{\mathbf{x}}$.
3. Form the matrix $A = [\mathbf{y}_1 \ \mathbf{y}_2 \ \cdots \mathbf{y}_n]$ ($d \times n$ matrix), then the covariance matrix can be computed

$$C = \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}_i \mathbf{y}_i^T = AA^T (d \times d \text{matrix})$$

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

1. $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i$.
2. Subtract the mean: $\mathbf{y}_i = \mathbf{x}_i - \bar{\mathbf{x}}$.
3. Form the matrix $A = [\mathbf{y}_1 \ \mathbf{y}_2 \ \cdots \mathbf{y}_n]$ ($d \times n$ matrix), then the covariance matrix can be computed

$$C = \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}_i \mathbf{y}_i^T = AA^T (d \times d \text{matrix})$$

4. Compute the eigenvalues of $C : \lambda_1 > \lambda_2 > \cdots > \lambda_d$, and corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_d$.

Methodology

Suppose $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ are $d \times 1$ vectors, then

1. $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i$.
2. Subtract the mean: $\mathbf{y}_i = \mathbf{x}_i - \bar{\mathbf{x}}$.
3. Form the matrix $A = [\mathbf{y}_1 \ \mathbf{y}_2 \ \cdots \mathbf{y}_n]$ ($d \times n$ matrix), then the covariance matrix can be computed
$$C = \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}_i \mathbf{y}_i^T = AA^T (d \times d \text{matrix})$$
4. Compute the eigenvalues of $C : \lambda_1 > \lambda_2 > \cdots > \lambda_d$, and corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_d$.
5. Keep only the terms corresponding to the k largest eigenvalues: $\widehat{\mathbf{x}} - \bar{\mathbf{x}} \approx \sum_{i=1}^{k} b_i \mathbf{u}_i$, i.e., the representation of $\widehat{\mathbf{x}} - \bar{\mathbf{x}}$ into the basis $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k$.

Information loss

Analysis

- $\widehat{\mathbf{x}} - \overline{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i$, i.e., $\widehat{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i + \overline{\mathbf{x}}$

Information loss

Analysis

- $\widehat{\mathbf{x}} - \overline{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i$, i.e., $\widehat{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i + \overline{\mathbf{x}}$
- It can be shown that the low-dimensional basis based on principal components minimizes the reconstruction error: $e = \|x - \widehat{x}\|_F$

Information loss

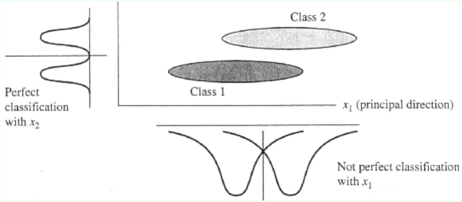
Analysis

- $\widehat{\mathbf{x}} - \overline{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i$, i.e., $\widehat{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i + \overline{\mathbf{x}}$
- It can be shown that the low-dimensional basis based on principal components minimizes the reconstruction error: $e = \|x - \widehat{x}\|_F$
- It can be shown that the error is equal to $e = \sqrt{\sum_{i=k+1}^{d} \lambda_i}$.

Information loss

Analysis

- $\widehat{\mathbf{x}} - \overline{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i$, i.e., $\widehat{\mathbf{x}} = \sum_{i=1}^{k} b_i \mathbf{u}_i + \overline{\mathbf{x}}$
- It can be shown that the low-dimensional basis based on principal components minimizes the reconstruction error: $e = \|x - \widehat{x}\|_F$
- It can be shown that the error is equal to $e = \sqrt{\sum_{i=k+1}^{d} \lambda_i}$.
- PCA is not always an optimal dimensionality-reduction procedure, e.g., classification problem.



SVD: Pros & Cons

Pros

- Optimal low-rank approximation in L_2 norm.

SVD: Pros & Cons

Pros

- Optimal low-rank approximation in L_2 norm.
- There are many implementations, such as LINPACK, Matlab, SPlus, Mathematica...

Cons

SVD: Pros & Cons

Pros

- Optimal low-rank approximation in L_2 norm.
- There are many implementations, such as LINPACK, Matlab, SPlus, Mathematica...

Cons

- Conventional SVD is undefined for incomplete matrices.

SVD: Pros & Cons

Pros

- Optimal low-rank approximation in L_2 norm.
- There are many implementations, such as LINPACK, Matlab, SPlus, Mathematica...

Cons

- Conventional SVD is undefined for incomplete matrices.
- The complexity of computing SVD is $O(nm^2)$ or $O(n^2m)$. Less work if we want first k singular vecotrs or matrix is sparse.

SVD: Pros & Cons

Pros

- Optimal low-rank approximation in L_2 norm.
- There are many implementations, such as LINPACK, Matlab, SPlus, Mathematica...

Cons

- Conventional SVD is undefined for incomplete matrices.
- The complexity of computing SVD is $O(nm^2)$ or $O(n^2m)$. Less work if we want first k singular vecotrs or matrix is sparse.
- We need an approach that can simply ignore missing values and reduce the complexity.

Take-home messages

- The Curse of Dimensionality
- Singular Value Decomposition
  - Diagonalization
  - Singular Value Decomposition
- Principal Component Analysis