Data Analysis

Dimensionality Reduction

Announcements

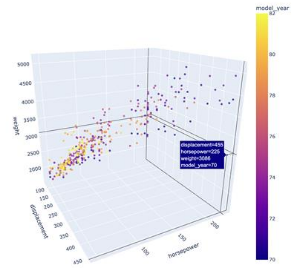
- We are almost there! 🎉
- Master Open Day tomorrow at PHS (with vlaai) 🍰
- Clinic 1 grading is half-way ☑️
- Clinic 2 grading will start soon ⏳
- Clinic 3 is due next week Friday (remaining wildcards to be used) 🃏
- Clinics 4 and 5 are not graded but count as "bootcamps" 🏋️
- Mid-course survey, fill it please 🔗
  https://forms.gle/nUAP8Lb4WmkQ62k36
- Last lecture on Monday, March 10th, pick the content 📦
  https://app.wooclap.com/UMDA
- Anything else? 🤔

Learning goals

- Discuss and justify the need for DR
- Interpret the rationale of SVD and PCA
- Analyze the result of SVD (singular values, U and V matrices)
- Analyze the result of PCA (PCs) in context
- Select the proper dimensionality reduction (rank, dimension)
- Derive the linear transformation process of PCA
- Assess the contributions of original variables to the PCs (loadings)
- Explain the outcome of PCA in terms of variance/axis transformation
- Apply PCA to datasets for DR and/or visualization
- Design & assess matrix factorizations from an optimization perspective

Visualization motivation

NOTEBOOK DEMO



Why dimensionality reduction?

We want visualize and get sanity out

- Visualization:
  Understand structure of the data
- Computational:
  Compress data → time/space efficiency
- Statistical:
  Remove redundancies
  Fewer dimensions → better generalization
- (Anomaly detection, Noise removal):
  Better detection of outliers/noise

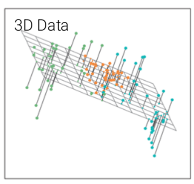
High-Level Objective of Dimensionality Reduction

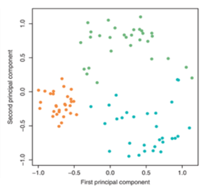
High-dimensional data might actually have a smaller **intrinsic dimension**

Dimensionality reduction: take **high-dimensional data** (*many columns*) and **find a smaller set** of **new features** (*columns*) that **approximately capture the information** in the original data.

Useful for **data visualization**, **EDA** and some **modeling tasks**.

Can be framed as a **matrix factorization** problem.





Dimensionality of Data?

Consider the dataset below.
We can think of the variables (*columns*)
of a dataset as its dimensionality.

Weight (lbs)	Weight (kg)
113.0	51.3
136.5	61.9
153.0	69.4

No Write or Wrong

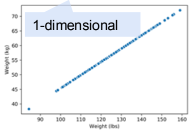
What would you call the columns space of this dataset?
A. 1-dimensional
B. 2-dimensional

Dimensionality of Data?

Consider the datasets shown.

Weight (lbs)	Weight (kg)
113.0	51.3
136.5	61.9
153.0	69.4

Height (in)	Weight (kg)	Weight (lbs)	Age
65.8	51.3	113.0	17
71.5	61.9	136.5	21
69.4	69.4	153.0	18



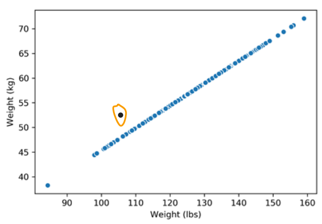
Rank of Matrix

- **3-dimensional**, because two weight columns are redundant.
- 🤓 notice: matrix of dataset has (column) **rank** 3, while the dataset on the left has **rank** 1

Dimensionality - what does it mean...?

Note that in the dataset below, I've added one **outlier** point to the 1d-dataset

- Just this one outlier is enough to change the rank of the matrix to 2.
- But the data is still ***approximately*** 1-dimensional!



Intrinsic Dimension of a dataset is the **minimal set of dimensions** needed to approximately represent the data.

Dimensionality reduction is generally an **approximation of the original data**. This is often achieved through matrix factorization.

Dimensionality Reduction as Matrix Factorization

Original Dataset

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75
630	31	2.58
124	24	2

≈

Reduced Dimension Dataset

Age (days)	Height (in)
182	28
399	30
725	33
630	31
124	24

*

	?	

We only keep one of the Colums

One **linear** technique to dimensionality reduction is via **matrix decomposition**, which is closely tied to **matrix multiplication**.

Market basket matrices

$$\begin{array}{cc} & \text{d products (e.g., milk, bread, wine, etc.)} \\ \text{n customers} & \begin{pmatrix} \\ A \\ \text{A}_{ij} = \text{quantity of j-th product purchased by the i-th customer} \end{pmatrix} \end{array}$$

Find a subset of the products that characterize customer behavior

Social-network matrices

$$\begin{array}{cc} & \text{d groups (e.g., news, sports, etc.)} \\ \text{n users} & \begin{pmatrix} \\ A \\ \text{A}_{ij} = \text{partiticipation of the i-th user in the j-th group} \end{pmatrix} \end{array}$$

Find a subset of the groups that accurately clusters social-network users

Document matrices

$$
\begin{array}{cc}
 & \textbf{d} \text{ words (or terms)} \\
 & \text{(e.g., theorem, proof, etc.)} \\
\textbf{n} \text{ documents} & \left(\begin{array}{c} A \\ A_{ij} = \text{frequency of the } \textbf{j}\text{-th term in the } \textbf{i}\text{-th document} \end{array} \right)
\end{array}
$$

Find a subset of the terms that accurately clusters the documents

Digital images

$$
\begin{array}{cc}
 & \textbf{d} \text{ pixels (if I order them, e.g. if image is 12x12, then } |p|=144 \\
\textbf{n} \text{ images} & \left(\begin{array}{c} A \\ A_{ij} = \text{pixel value of } \textbf{j}\text{-th pixel for image } \textbf{i} \end{array} \right)
\end{array}
$$

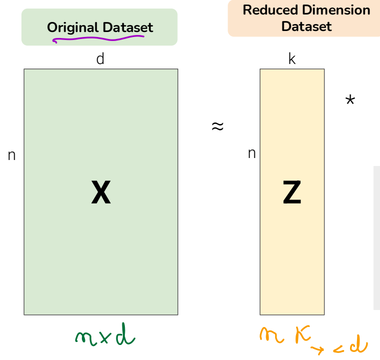
Find which pixels are more important across all images

Netflix data

$$
\begin{array}{cc}
 & \textbf{d} \text{ titles} \\
\textbf{n} \text{ viewers} & \left(\begin{array}{c} A \\ A_{ij} = \text{binary value, 1 is the } \textbf{i}\text{-th viewer has watched the } \textbf{j}\text{-th title} \end{array} \right)
\end{array}
$$

Find a subset of the movies that accurately describe the behavior or the viewers

Dimensionality Reduction as Matrix Factorization



We wan to develop a procedure to **decompose our data matrix X** into a **lower dimensions matrix Z** that when multiplied by **W approximately recovers the original data**.

Interpreting Matrix multiplication

Consider the matrix multiplication example below.

- Each row of the fruits matrix represents one bowl of fruit.
  - First bowl: 2 apples, 2 lemons, 2 melons.
- Each column of the dollars matrix represents the cost of fruit at a store.
  - First store: 2 dollars for an apple, 1 dollar for a lemon, 4 dollars for a melon.
- Output is the cost of each bowl at each store.

Number of Fruits

	# Apples	# Lemons	# Melons
Bowl 1	2	2	2
Bowl 2	5	8	0

X

Fruit Costs

	Store 1	Store 2
$/Apple	2	1
$/Lemon	1	1
$/Melon	4	1

matmul =

Output

	Store 1	Store 2
Bowl 1	14	6
Bowl 2	18	13

2×3 3×2

Two ways to **interpret** matrix multiplication:
1. Linear operations per $\boxed{\text{datapoint}}$
2. Column transformation.

Matrix Decomposition (Matrix Factorization)

Matrix decomposition (a.k.a. Matrix Factorization) is the opposite of matrix multiplication, i.e. taking a matrix and decomposing it into two separate matrices.

- Just like with real numbers, there are infinitely many such decompositions.
  - 9.9 = 1.1 * 9 = 3.3 * 3.3 = 1 * 9.9 = ...
- Matrix sizes aren't even unique...

Get here

We want start from here

Some example factorizations:

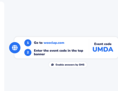
$$
3\text{x}2 \quad
\begin{array}{cc}
\begin{bmatrix} 182 & 28 \\ 399 & 30 \\ 725 & 33 \end{bmatrix} \\
\text{Age} \quad \text{Height (in)}
\end{array}
\times
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1/12 \end{bmatrix} \quad 2\text{x}3
$$

we don't have it

Goal   Transformation

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

Matrix Decomposition: Infinite ways?



Some example factorizations:

3x2

182	28
399	30
725	33

X

2x3

1	0	0
0	1	1/12

3x3

182	28	2.33
399	30	2.5
725	33	2.75

X

3x3

1	0	0
0	1	0
0	0	1

I

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

What are **possible** matrix factorizations? Select all that apply.
A. (3x2) x (2x3) ✓ **C.** (3x1) x (1x3) **E.** (3x🦄) x (🦄x3)
B. (3x3) x (3x3) ✓ **D.** (3x4) x (4x3) ✓

Matrix Decomposition: Limited by Rank

3x4

182	28	2.33	0
399	30	2.5	0
725	33	2.75	0

X

4x3

1	0	0
0	1	0
0	0	1
99	31	17

Fine, but defeats the point of dimension **reduction**...

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

Impossible, because rank of original > 1!

ax/bx = a/b = 182/399
ay/by = a/b = 28/30
Contradiction!

3x1

a
b
c

X

1x3

x	y	z

In practice we usually construct decompositions < rank of the original matrix!

They provide **approximate reconstructions** of the original matrix.

Automatic and Approximate factorization

A rank R matrix can be decomposed into an R dimensional representation x (times) some transformation matrix.

But what if we wanted a lower-dimensionality?

In the example below, the rank of the 4D matrix is 3, so we can no longer exactly reconstruct the 4-D matrix. Still, some 2D matrices yield better approximations than others. How well can we do?

Rank 3 → Not a linear Combo

100 x 4

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

≈

100 x 2

...	...

X

2 x 4

Principal Component Analysis (PCA)

Goal: Transform observations from high-dimensional data down to low dimensions (e.g. 2) through **linear transformations.**

Related Goal: Low-dimension representation should capture (as much as possible) of the variance of the original data.

100 x 4

width	length	area	perimeter
20	20	400	80
16	12	192	56
10	10	100	40
...	...	...	...
24	12	288	72

X

Principal Components (cols)

4x2

First PC	Second PC

Transformation Matrix

Latent Factors (cols)

100 x 2

...	...

Two Equivalent Framings of PCA

There are two equivalent ways to frame PCA:

1. Finding the directions of **maximum** variance in the data in the low dim
2. Finding the low dimensional (rank) matrix factorization that **best approximates the data**

We will focus on the first one, aka **variance maximization** framing (more common) and then return to the **best approximation** framing (more general).

The second framing allows the problem to be done like a (normal) optimization problem

Capturing Total Variance

We define the total variance of a data matrix as the sum of variances of attributes. of my Variables

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

Total Variance: 402.56 = 7.69 5.35 338.73 50.79

Depends on the Unit

Might change based on the scaling

Goal of PCA, restated:

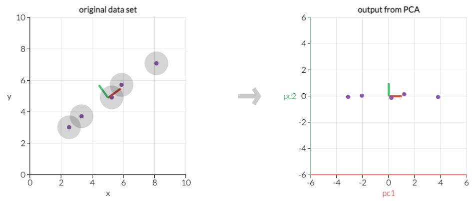
Find a linear transformation that creates a low-dimension representation which captures as much of the original data's total variance as possible.

Let’s derive PCA, in all gory math 🔢🩸💀

NOTEBOOK

Cov Matrix

Eigen val



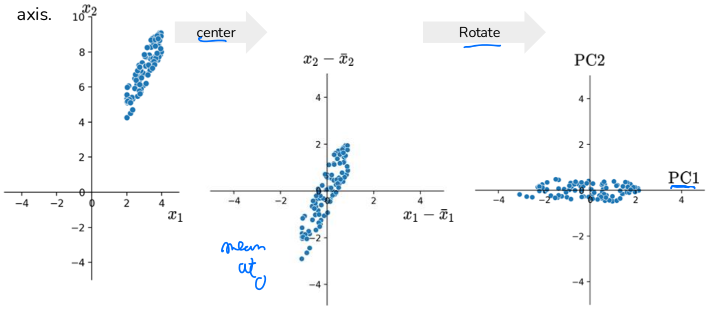
PCA "Uncorrelates the Data" https://setosa.io/ev/principal-component-analysis/

Scree P After Break

How PCA Transforms Data, Visually

Back Prop

PCA first centers the data matrix, then rotates it such that the direction with the most variance (i.e. the direction that's the most spread-out) is aligned with the x-axis.



Important Note on Centering and Standardizing the Data

When running PCA it is important to **center the data** (ensure data is centered at 0):

```
X = X - np.mean(X, axis = 0)
```

- The scikit-Learn PCA code does this automatically

You should also *consider* **standardizing your data** (ensure **unit variance** on each dimension)

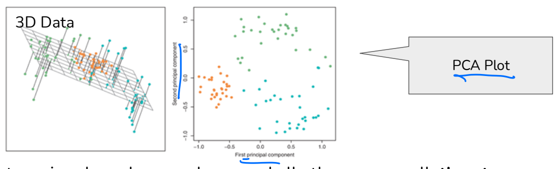
```
X = (X - np.mean(X, axis = 0))  / np.std(X, axis = 0)
```

- The scikit-Learn PCA code **does not do this**
- You **should standardize your data** if features have **vastly different ranges**
- You should not standardize your data if the units are all the same and the differences in **variability across dimensions is meaningful** (e.g., everything is in the same units)

PCA Plot

We often construct a scatter plot of the data projected onto the **first two principal components**. This is often called a PCA plot.

- PCA plots allow us to visually assess similarities between our data points and if there are any clusters in our dataset.

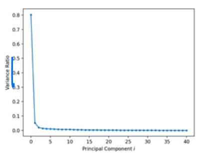


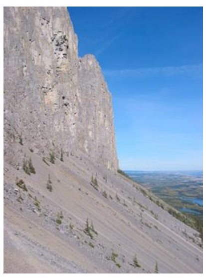
If the first two singular values are large and all others are small, **then two dimensions are enough to describe most of what distinguishes one observation from another.** If not, then a PCA plot is omitting lots of information.

Scree Plot

If the first two singular values are large and all others are small, then **two dimensions are enough** to describe most of what distinguishes one observation from another. If not, then a PCA scatter plot is omitting lots of information.

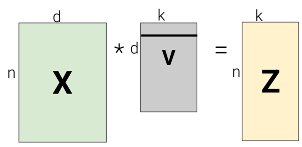
A scree plot shows the variance ratio captured by each principal component, largest first.





Scree [wikipedia]

Biplot



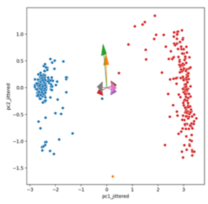
The *i*-th row of ***V*** indicates **how feature *i* contributes** to each PC.

Biplots superimpose the directions onto the plot of PC1 vs. PC2.

Row i of V corresponds to the direction for feature i, e.g., (v_{i1}, v_{i2}).

- There are several ways to scale biplots vectors

Through biplots, we can interpret how features correlate with the principal components shown: positively, negatively, or not much at all.

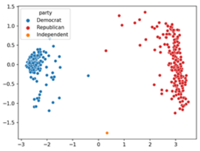


Summary of PCA plots

PCA Plot

Scatter plot of PC1 against PC2

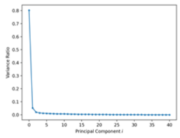
- Helps us assess similarities between our data points and if there are any clusters in our dataset.



Scree Plot

Line plot showing the variance ratio captured by each principal component, largest first.

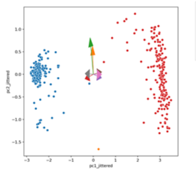
- If first two is large enough, we know PCA plot is good representation of data
- "Elbow" method to assess how many PCs to use



Biplot

PCA plot + directions of feature importance for PC1 and PC2

- All benefits from PCA plot, and
- Shows how much some features contribute to PC1/2



Derive PCA using Loss Minimization

Goal: Minimize the **reconstruction loss** for our **matrix factorization model**:

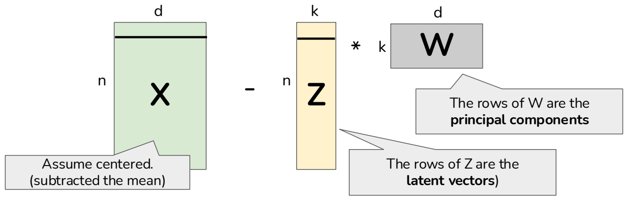
$$L(Z, W) = \frac{1}{n} \sum_{i=1}^{n} \| X_i - Z_i W \|^2$$

What we want

PCA task

Row Vector

Row Vector



Singular Value Decomposition

We can use SVD to do PCA

Singular value decomposition (SVD) describes a matrix decomposition into 3 matrices:

$$X = U \quad S \quad V^T$$

$X \in \mathbb{R}^{n \times d}$

$U \in \mathbb{R}^{n \times d}$ — columns of U are orthonormal. Columns of U are **eigenvectors of XX^T**

$S \in \mathbb{R}^{d \times d}$ — diagonal matrix of singular values, ordered from largest to smallest r non-zero singular values. rank r ≤ d

$V \in \mathbb{R}^{d \times d}$ — columns of V are orthonormal. Columns of V are eigenvectors of X^TX

*note 1: assume d < n.

There are infinite possible factorizations! SVD chooses a special (but non-unique) one with these properties.

SVD one-by-one

$$U$$

$U \in \mathbb{R}^{n \times d}$

- Columns of U are orthonormal: $\vec{u}_i^\top \vec{u}_j = 0$ for all *i*, *j* and all vectors are unit vectors
- Columns of U are called the left singular vectors (aka are the eigenvectors of XX^T)
- $UU^T = I_n$ and $U^TU = I_d$
- Can think of U as a rotation

Eigenvectors of XX^T:

$$XX^T = USV^T (USV^T)^T = USV^TVS^TU^T$$
$$= USS^TU^T$$

SVD one-by-one

$$S$$

$S \in \mathbb{R}^{d \times d}$

- Diagonal values (singular values), are ordered from greatest to least
- r non-zero singular values

$$\begin{array}{c} \\ d \end{array}\left[\begin{array}{cccc|ccc} s_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{array}\right]$$

Rank

d

- r is the rank of X
- The majority of the matrix is zero
- The singular values $s_1, s_2, \ldots, s_r$ are **non-negative** and **sorted** $s_1 \geq s_2 \geq \ldots \geq s_r$
- Can think of S as a scaling operation

SVD one-by-one

$$V^\top$$

$V \in \mathbb{R}^{d \times d}$

- Columns of V are orthonormal → rows of V^T are orthonormal
- Columns of V are called the right singular vectors (aka are the eigenvectors of X^TX)
- $VV^T = V^TV = I_d$
- Can think of V as a rotation

Eigenvectors of X^TX:

$$X^TX = (USV^T)^T USV^T = VS^TU^TUSV^T$$
$$X^TX = VS^TSV^T$$

NumPy SVD

```
U, S, Vt = np.linalg.svd(X, full_matrices = False)
```

$\underbrace{\texttt{full\_matrices = False}}_{U \in \mathbb{R}^{n \times d}}$

[documentation]

$$X = U \quad S \quad V^T$$

width	height	area	Perim.
2.97	1.35	24.78	8.64
-3.03	-0.65	-15.22	-7.36
-4.03	-1.65	-20.22	-11.36
3.97	-1.65	3.78	4.64
3.97	3.35	48.78	14.64
-2.03	-3.65	-20.22	-11.36
-1.03	-2.65	-15.22	-7.36
0.97	0.35	6.78	2.64
1.97	-3.65	-16.22	-3.36
2.97	-2.65	-7.22	0.64
...	...	...	...

-0.13	0.01	0.03	-0.21
0.09	-0.08	0.01	0.56
0.12	-0.13	0.09	-0.07
-0.03	0.18	0.01	-0.05
-0.26	-0.09	0.09	-0.06
0.12	-0.05	0.17	-0.05
0.09	0	0.1	-0.08
-0.04	0.01	0	-0.08
0.08	0.18	0.04	-0.05
0.03	0.19	0.02	-0.05
...	...	...	...

197.39			
	27.43		
		23.26	
			0

-0.1	-0.07	-0.93	-0.34
0.67	-0.37	-0.26	0.59
0.31	-0.64	0.26	-0.65
0.67	0.67	0	-0.33

X is therefore **rank 3**.

Principal Components are the Eigenvectors of the Covariance Matrix!

Assume we have constructed the Singular Value Decomposition (SVD) of X:

$$X = USV^T$$

Because **X is centered** the covariance matrix **of X** is:

$$\Sigma = X^T X = \left(USV^T\right)^T USV^T = VS^T \underbrace{U^T U}_{I} SV^T$$
$$= VS^2V^T$$

Right multiplying both sides by V we get:

The columns of V are the eigenvectors of the covariance matrix Σ and therefore the Principal Components

$$\Sigma V = VS^2V^TV = VS^2$$

The squared **singular values** are the **eigenvalues** of Σ

Variance and Singular Values

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72
7.69	5.35	338.73	50.79

Total Variance: **402.56** =

Formally, the ith singular value tells us the **component score**, i.e., how much of the variance is captured by the ith principal component. N is # of datapoints.

$$\textbf{i-th component score} = \frac{\textbf{(i-th singular value)}^2}{\textbf{N}}$$

197.4	0	0	0
0	27.43	0	0
0	0	23.26	0
0	0	0	0

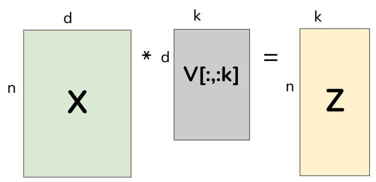
→ 197.39²/100=389.63
→ 27.43²/100 = 7.52
→ 23.26² / 100= 5.41
Sum = **402.56**. ✅

From SVD to PCA

We have now shown that if we construct the singular value decomposition of X:

$$X = USV^T$$

The **first k** columns of V are the **first k principal components** and we can construct the **latent vector** representation of X by projecting X onto the **principal components**



This gives us another way to do PCA!

$$Z = XV = USV^TV$$
$$= US$$

Using only the **first k** columns and **rows** of **U** and **S**

Computing Latent Vectors Using X * V

Constructing a 2 principal component approximation (k=2)

$$X \quad * \quad V \quad = \quad Z$$

width	height	area	Perim.
2.97	1.35	24.78	8.64
-3.03	-0.65	-15.22	-7.36
-4.03	-1.65	-20.22	-11.36
3.97	-1.65	3.78	4.64
3.97	3.35	48.78	14.64
-2.03	-3.65	-20.22	-11.36
...	...	...	...

PC1	PC2		
-0.1	0.67	0.31	0.67
-0.07	-0.37	-0.64	0.67
-0.93	-0.26	0.26	0
-0.34	0.59	-0.65	-0.33

-26.43	0.16
17.05	-2.18
23.25	-3.54
-5.38	5.03
-51.09	-2.59
23.19	-1.45

Computing Latent Vectors Using U * S

Constructing a 2 principal component approximation (k=2)

$$U \quad * \quad S \quad = \quad Z$$

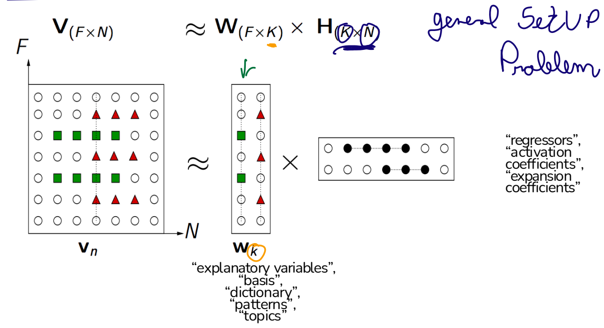
-0.13	0.01	0.03	-0.21
0.09	-0.08	0.01	0.56
0.12	-0.13	0.09	-0.07
-0.03	0.18	0.01	-0.05
-0.26	-0.09	0.09	-0.06
0.12	-0.05	0.17	-0.05
...	...	...	...

197.39			
	27.43		
		23.26	
			0

-26.43	0.16
17.05	-2.18
23.25	-3.54
-5.38	5.03
-51.09	-2.59
23.19	-1.45

Same Result

Matrix factorization (in general)



A popular variant: Non-Negative MF

- Some entries in our matrices are non-negative
  - Think about ratings (or even our rectangle data)
  - Factorization is an algebraic operation meaning that getting non-negative entries does not help us understand what is going on in the data
  - We would like to encode prior knowledge that latent dimensions are non-negative
- *Also*: We would like the effect of factors to be accumulative (i.e. no cancellations due to negative contributions)
- *Also*: Standard SVD assumes missing entries to be zero

NMF example – Recommender system

- Given a NXM matrix R with some entries unknown
  - N columns represent items
  - M rows represents users
  - Entry R_{ij} represents the i-th user's rating of j-th item
- We are interested in the unknown entries' possible values & the hidden dimensions
- Here I will attempt:

$R_{N\times M} \approx W_{N\times 2} \times H_{2\times M}$

	Passengers	War of the Worlds	Bride Wars	The Matrix R
John	2		3	
Arya	0	5	0	4
Sansa	4			5
Robb	4	5		

NMF example – Recommender system

- Problem is that there are many missing entries but NMF handles this problem!

note: the default NMF from sklearn does not handle missing values

	L1	L2
John	0.2	0
Arya	0	0.5
Sansa	0.5	0
Robb	0	0.5

	Passengers	War of the Worlds	Bride Wars	The Matrix Reloaded
L1	6.3	0	1.1	8
L2	3.9	10.7	0	3.3

"Gods" of the Men

NMF example – Recommender system

- Estimate unknown ratings as inner products of latent factors
  - Rating of Arya for Passengers: 6.3*0 + 3.9*0.5 = 1.95

John	2		3	
Arya		5		4
Sansa	4			5
Robb	4	5		

≈

Passengers	War of the Worlds	Bride Wars	The Matrix Reloaded
1.3	0	0.2	1.6
2	5.4	0	1.7
3.2	0	0.6	4
2	5.4	0	1.7

NMF as an optimization problem

- Minimize:

$$\|V - WH\|^2$$

No direct Algebra Solution

subject to the constraints W>0, H>0

- Possible numerical methods exist:
  - Coordinate descent: Fix W and optimize H, fix H and optimize W until the tolerance threshold is met

    Step-wise
  - Multiplicative Update using the following steps (Lee & Seung), 2001) so as to guarantee non-negative entries:

$$H_{a\mu} \leftarrow H_{a\mu}\frac{\sum_i W_{ia}V_{i\mu}/(WH)_{i\mu}}{\sum_k W_{ka}} \qquad W_{ia} \leftarrow W_{ia}\frac{\sum_\mu H_{a\mu}V_{i\mu}/(WH)_{i\mu}}{\sum_\nu H_{a\nu}}$$

Cool applications of MF

PCA used for denoising images

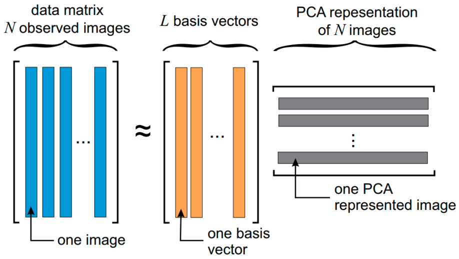
Denoised image using 15 PCA components

- I compute the PCA of the image
- I "reconstruct" the image using only 15 PCs

PCA: Eigenfaces

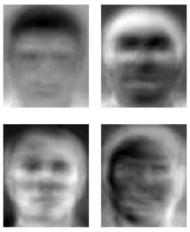
- Eigenfaces are
  the eigenvectors of
  the covariance matrix of
  the probability distribution of
  the vector space of
  human faces
- Eigenfaces are the 'standardized face ingredients' derived from the statistical analysis of many pictures of human faces
- A human face may be considered to be a combination of these standard faces

How does it work?



Eigenfaces (cont.)

- the principal eigenface looks like a bland androgynous average human face



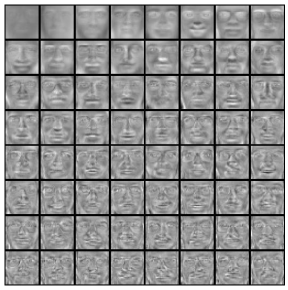
http://en.wikipedia.org/wiki/Image:Eigenfaces.png

Eigenfaces – Steps 1,2

<u>Input:</u>
- Dataset of N face images
  - Assume each image is KxK
  - Represent this as a K^2 x N matrix

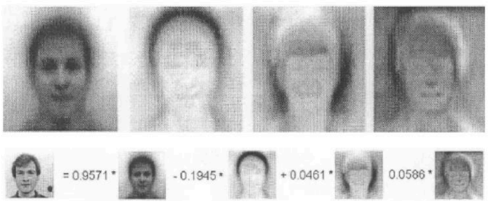
Eigenfaces – Step 3

- Apply PCA and keep only the m significant dimensions
- Fold back into KxK images



Eigenfaces – Step 4

- Represent faces onto this basis
  - Using the best K eigenvectors



Eigenfaces – Face Recognition

- When properly weighted, eigenfaces can be summed together to create an approximate gray-scale rendering of a human face.
- Remarkably few eigenvector terms are needed to give a fair likeness of most people's faces
- Hence, eigenfaces provide a means of applying <u>data compression</u> to faces for identification purposes.

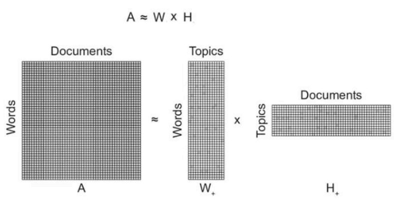
Mostly Insensitive to lighting expression, orientation

Eigenfaces – Face Recognition

- What happens when I project a new face?

- Remember the the eigenfaces is the “new representation basis”, hence everything has to be represented based on these

NMF for text topic modeling



NMF for audio source separation

http://d-kitamura.net/demo-defNMF_en.html