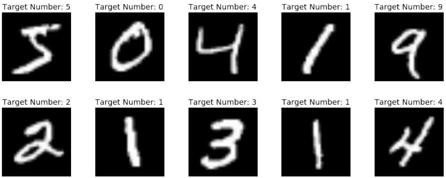
# Intro Principle Component Analysis

Frederik Mallmann-Trenn
6CCS3AIN

# Let's say you want to recognise digits

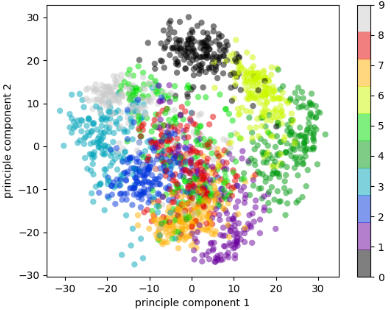

- MNIST: Very famous dataset from scikit-learn
- Let's say you want to use the large training set with examples (128x128 pixels)
- So that when I draw you a new digit, you can tell what it is! 

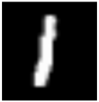
# Let's say you want to recognise digits



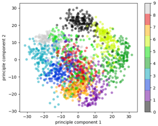

- Problem: Each digit has $128 \cdot 128 = 16,384$ features/dimensions
- Is there a nice way to reduce the number of features/dimensions?

# A cool way of doing this

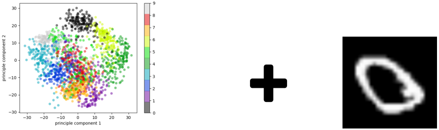


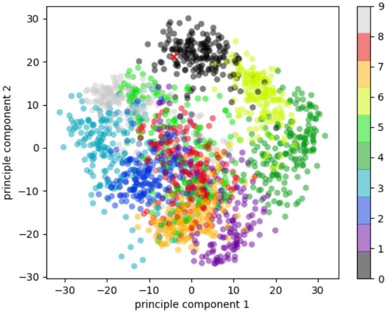

- We try to find the two components (each is a combination of the features) 

# PCA




# PCA






- Red cross is new input
- Easy to figure out where it belongs to..

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)
- Strong theoretical background

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)
- Strong theoretical background
- Can also be used to store data in more efficient way (Image compression).

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)
- Strong theoretical background
- Can also be used to store data in more efficient way (Image compression).
- Visual evaluation possible for a small number of components (say 2)

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)
- Strong theoretical background
- Can also be used to store data in more efficient way (Image compression).
- Visual evaluation possible for a small number of components (say 2)

# PCA / SVD

Advantages

- State-of-the-art for many applications (supervised and unsupervised)
- Incredibly efficient (often, almost linear time)
- Strong theoretical background
- Can also be used to store data in more efficient way (Image compression).
- Visual evaluation possible for a small number of components (say 2)

Small disclaimer: PCA and SVD (Singular value decomposition) are slightly different, but very very similar, we'll look at PCA (which often uses SVD)

# Example Application Principle Component Analysis

Frederik Mallmann-Trenn
6CCS3AIN

# PCA Example

- Say you have a bunch of house listings and you would like to group them into student housing, regular and luxury

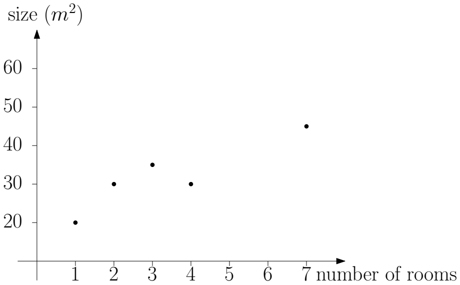
# PCA Example

- Let's say we have the following features
  - Floor size ($m^2$)
  - Number of rooms
  - Distance supermarket
  - Distance King's
  - Hipster vibe
- Let's say we want to reduce to two features to have a nice visual representation.
- Can we reduce it to two features?

# Why does PCA work?

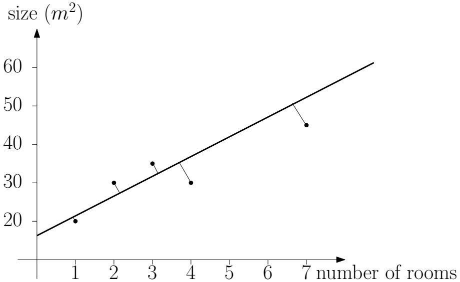
- Reduce to two or three features
  - Size
    - Floor size ($m^2$)
    - Number of rooms
  - Location
    - Distance supermarket
    - Distance King's
    - Hipster vibe
- Why does this make sense?

# PCA



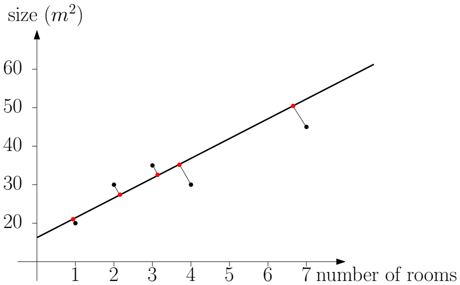

- For example the floor size and the number of rooms are often correlated
- Let's see how it would look like if we compressed both dimensions to one dimension

# PCA



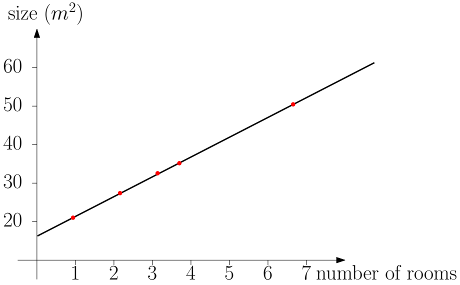

- If we take the line that minimises the Least Squares Distance, we get ...

# PCA



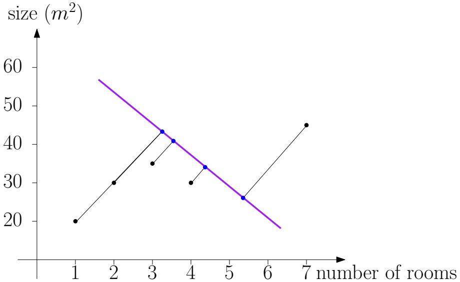

- If we take the line that minimises the Least Squares Distance, we get ...
- ... the following projection of the points.

# PCA



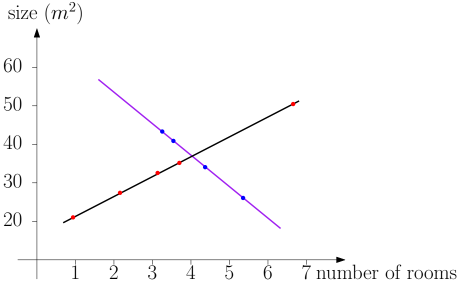

- After cleaning up, this is what we get

# PCA



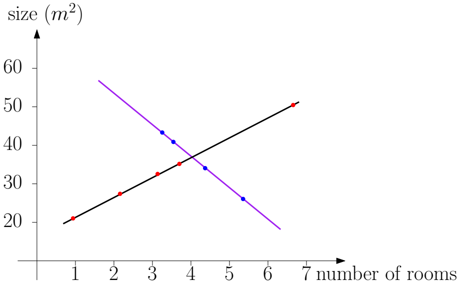

- What if we take a different line? (purple)?

# PCA



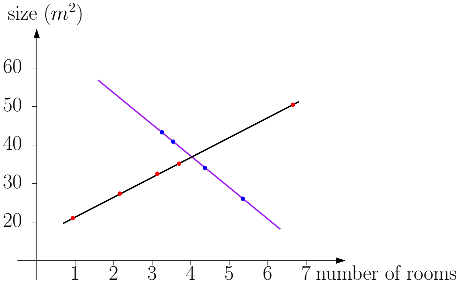

- The spread here is the variance of the data
- And we would like to maximise it.

# PCA



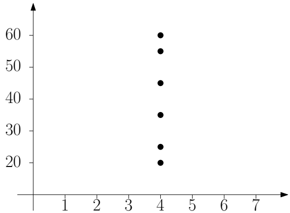

- The spread here is the variance of the data
- And we would like to maximise it.
- Intuitively, the more variance we capture, the better we can approximate the higher-dimensional space (here $d = 2$)

# PCA

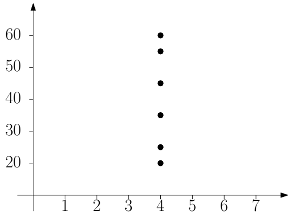


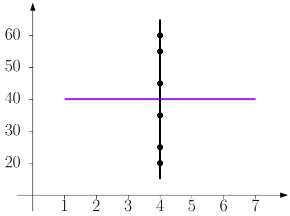

- The spread here is the variance of the data
- And we would like to maximise it.
- Intuitively, the more variance we capture, the better we can approximate the higher-dimensional space (here $d = 2$)
- If we compare them, we see that the points are less spread out on the purple line
- The black line actually maximises the spread and therefore is the best for approximating the higher-dimensional space

# Why should we maximise the variance?



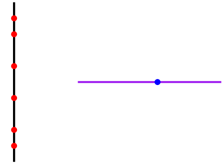
# Why should we maximise the variance?



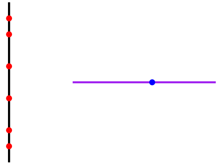

- Left: New input
- Right: Two potential lines onto which we can project.
- Consider projecting to a **horizontal** and a **vertical** line.

# Why should we maximise the variance?



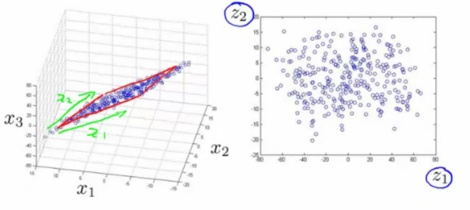
- This is how the output would looks like
- Which line retains more information?

# Why should we maximise the variance?



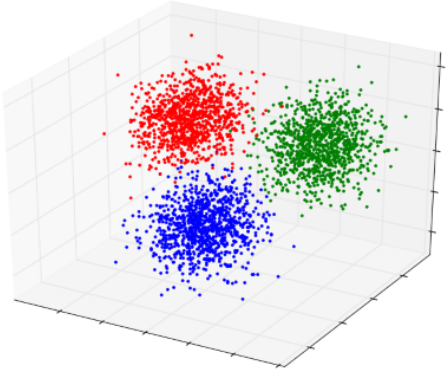
- This is how the output would looks like
- Which line retains more information?
- Clearly the black line, all points on the purple line are at the same location.

# 3D to 2D



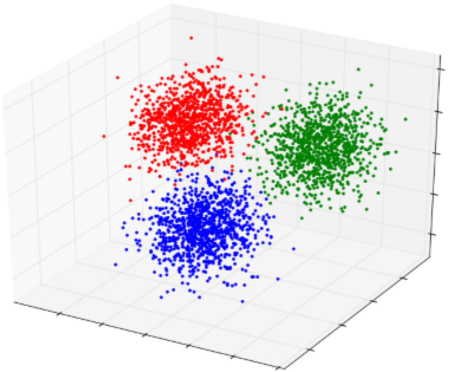

- Let's say our three dimensions ($x_1$, $x_2$, and $x_3$) are as on the l.h.s.
  - Distance supermarket
  - Distance King's
  - Hipster vibe
- Then after reducing it to $2D$ it looks like the r.h.s.
- We may also wish to reduce it to just a line $(1D)$, but we can see that this would be very lossy

# 5D to 3



- If we plot our 5D data using the components we found (1 for size and 2 for location)
- We get this 3D plot
- We can see that our different classes student housing, regular and luxury are well-separated.

# 5D to 3



- If we plot our 5D data using the components we found (1 for size and 2 for location)
- We get this 3D plot
- We can see that our different classes student housing, regular and luxury are well-separated.
- This is the whole point: reduce the information, but keep the important information!

# Matrix Multiplication

Frederik Mallmann-Trenn
6CCS3AIN

# Matrix

$$\begin{array}{c|cccc} & 1 & 2 & \dots & n \\ \hline 1 & & & & \\ 2 & & & & \\ 3 & & & & \\ \vdots & & & & \\ m & & & & \end{array}$$

$$\begin{array}{cc} & \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

- Here we have a $m \times n$ ($m$ by $n$) matrix.
- (image source: wikipedia)

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{1}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{1}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise
- Formula $c_{i,j} = \sum_k a_{i,k} \cdot b_{k,j}$

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{1}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise
- Formula $c_{i,j} = \sum_k a_{i,k} \cdot b_{k,j}$
- $c_{1,1} = a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} + a_{1,3} \cdot b_{3,1}$

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{2}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{2}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise
- Formula $c_{i,j} = \sum_k a_{i,k} \cdot b_{k,j}$

# Matrix Multiplication

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \tag{2}$$

- From left matrix: select matching row
- From right matrix: select matching column
- Multiply them component-wise
- Formula $c_{i,j} = \sum_k a_{i,k} \cdot b_{k,j}$
- $c_{2,3} = a_{2,1} \cdot b_{1,3} + a_{2,2} \cdot b_{2,3} + a_{2,3} \cdot b_{3,3}$

# Matrix Multiplication: example

- What happens if you multiply

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

# Matrix Multiplication: example

- What happens if you multiply

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

- First note that if you multiply a $m \times k$ matrix with $k \times n$ matrix, then the out outcome is $m \times n$.

# Matrix Multiplication: example

- What happens if you multiply

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

- First note that if you multiply a $m \times k$ matrix with $k \times n$ matrix, then the out outcome is $m \times n$.
- In this case, $5 \times 4$ since $m = 5, k = 1, n = 4$.

# Matrix Multiplication: example

- What happens if you multiply

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

- First note that if you multiply a $m \times k$ matrix with $k \times n$ matrix, then the out outcome is $m \times n$.
- In this case, $5 \times 4$ since $m = 5, k = 1, n = 4$.
- The outcome is

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

# Transpose

- The transpose of a matrix $A$ is an operation which flips the matrix along its diagonal (switches the row and column indices of the matrix)

$$\mathbf{A} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \end{pmatrix} \tag{5}$$

- becomes

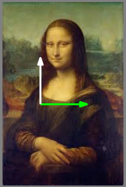
$$\mathbf{A}^T = \begin{pmatrix} a & e \\ b & f \\ c & g \\ d & h \end{pmatrix} \tag{6}$$

- Note that $\left(\mathbf{A}^T\right)^T = \mathbf{A}$

# Rotations and Eigenstuff

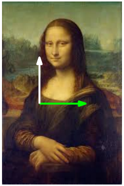
Frederik Mallmann-Trenn
6CCS3AIN

# Transformation Matrix



- Let's say our 'data' is the Mona Lisa
- Our base vectors are $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (in green) and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

# Transformation Matrix

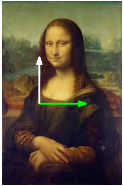


original

transformed

- We can stretch along the $y$-axis and squish along the $x$-axis with the matrix $\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix}$

# Transformation Matrix

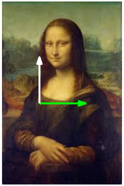


original

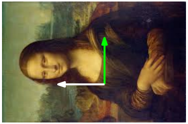
transformed

- We can stretch along the $y$-axis and squish along the $x$-axis with the matrix $\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix}$
- To see this, calculate $\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5x \\ 2y \end{pmatrix}$

# Transformation Matrix



original

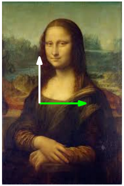


transformed

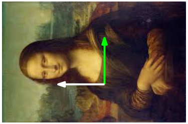
- We rotate counterclockwise

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

# Transformation Matrix



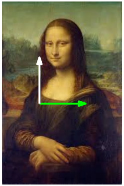
original



transformed

- We rotate counterclockwise
$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$
- To see this, calculate $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -y \\ x \end{pmatrix}$

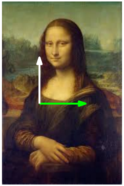
# Transformation Matrix



original transformed

- We can also perform a so-called shear mapping $\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$. Here $m \approx 1$.

# Transformation Matrix

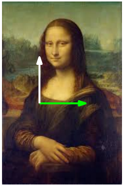


original

transformed

- We can also perform a so-called shear mapping $\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$. Here $m \approx 1$.
- To see this, calculate $\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + my \\ y \end{pmatrix}$

# Transformation Matrix

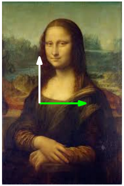


original

transformed

- Notice what happened to the vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$? Nothing.

# Transformation Matrix

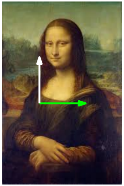


original

transformed

- Notice what happened to the vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$? Nothing.
- When a vector doesn't change its direction after multiplying with a matrix, then it's an eigenvector.

# Transformation Matrix

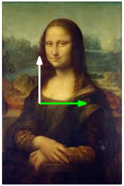


original

transformed

- In our stretching example from earlier, both vectors were actually eigenvectors.
- They didn't change the direction.
- However, they change the length.
- The value by which is changed the length is called the eigenvalue $\lambda$.

# Transformation Matrix

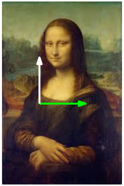


original

transformed

- The largest eigenvalue is $\lambda_1 = 2$ and the second largest here is $\lambda_2 = 0.5$.
- The corresponding eigenvectors are $\mathbf{v}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $\mathbf{v}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

# Transformation Matrix

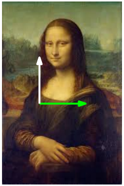


original

transformed

- What is an eigenvector for the matrix $\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$?

# Transformation Matrix



original transformed

- What is an eigenvector for the matrix $\begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}$?
- We can see that $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ must be one. What's the formula?

# Formula

- A vector $\mathbf{v}$ is an eigenvector of the matrix $\mathbf{M}$ if

$$\mathbf{M} \cdot \mathbf{v} = \lambda \mathbf{v}.$$

- $\lambda$ is the corresponding eigenvalue.

# Formula

- A vector $\mathbf{v}$ is an eigenvector of the matrix $\mathbf{M}$ if

$$\mathbf{M} \cdot \mathbf{v} = \lambda \mathbf{v}.$$

- $\lambda$ is the corresponding eigenvalue.
- Consider $\mathbf{M} = \begin{pmatrix} 2 & 2 \\ 5 & -1 \end{pmatrix}$.
- We can verify that $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\lambda_1 = 4$:

$$\mathbf{M} \cdot \mathbf{v}_1 = \begin{pmatrix} 2 & 2 \\ 5 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} = 4 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \lambda_1 \mathbf{v}_1$$

- The second eigenvector is $\mathbf{v}_2 = \begin{pmatrix} -2 \\ 5 \end{pmatrix}$ what's $\lambda_2$?

# PCA Algorithm

Frederik Mallmann-Trenn
6CCS3AIN

# PCA Algorithm - Data

- Let's say this is our data matrix (say our houses), where each data point is an $d$-dimensional row vector.
- $$\mathbf{X} = \begin{pmatrix} — & \mathbf{x}_1^T & — \\ — & \mathbf{x}_2^T & — \\ & \vdots & \\ — & \mathbf{x}_n^T & — \end{pmatrix}$$

  The dimensions are $n \times d$

# PCA Algorithm

- Step 1: Compute the mean row vector $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- Step 2: Compute the mean row matrix $\bar{\mathbf{X}} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \cdot \bar{\mathbf{x}}^T = \begin{pmatrix} — & \bar{\mathbf{x}}^T & — \\ — & \bar{\mathbf{x}}^T & — \\ & \vdots & \\ — & \bar{\mathbf{x}}^T & — \end{pmatrix}$

  The dimensions are $n \times d$

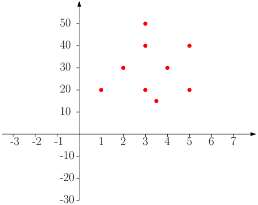
# PCA Algorithm

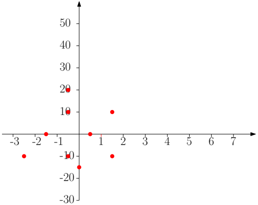
- Step 3: Subtract mean (obtain mean centred data)

$$B = \mathbf{X} - \bar{\mathbf{X}}$$

  The dimensions are $n \times d$

- Example:



# PCA Algorithm

- Step 4: Compute the covariance matrix of rows of $B$

$$\mathbf{C} = \mathbf{B}^T\mathbf{B}$$

The dimensions are $(n \times d)^T \times (n \times d) = (d \times n) \times (n \times d) = d \times d$

# PCA Algorithm

- Step 5: Compute the $k$ largest eigenvectors $\mathbf{v}_1, \mathbf{v}_1, \ldots, \mathbf{v}_k$ of $\mathbf{C}$ (not covered how to do this in this module. You use Python or WolframAlpha).
  Each eigenvector has dimensions $1 \times d$

  Pro tip: Python doesn't sort the eigenvectors for you. Sort eigenvectors by decreasing order of eigenvalues.

- Step 6: Compute matrix $\mathbf{W}$ of $k$-largest eigenvectors

$$\mathbf{W} = \begin{pmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_k \\ | & | & & | \end{pmatrix}$$

Dimensions of $\mathbf{W}$ are $(d \times k)$.

# PCA Algorithm

- Step 7: Multiply each datapoint $\mathbf{x}_i$ for $i \in \{1, 2, \ldots, n\}$ with $\mathbf{W}^T$

$$\mathbf{y}_i = \mathbf{W}^T \cdot \mathbf{x}_i$$

Dimensions of $\mathbf{y}_i$ are $(k \times d) \times (d \times 1) = k \times 1$

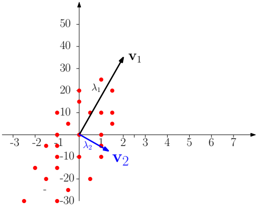
# PCA Algorithm

- Step 7: Multiply each datapoint $\mathbf{x}_i$ for $i \in \{1, 2, \ldots, n\}$ with $\mathbf{W}^T$

$$\mathbf{y}_i = \mathbf{W}^T \cdot \mathbf{x}_i$$

  Dimensions of $\mathbf{y}_i$ are $(k \times d) \times (d \times 1) = k \times 1$
- Congratulations! You've reduced the number of dimensions from $d$ to $k$!

# Why do we compute the covariance matrix?

- Example illustration: 

- The covariance matrix measures the correlation between pairs of features.
- Finding the largest eigenvectors allows us to explain most of the variance in data
- The more variance is explained by the eigenvectors, the more important they are

# Why do we compute the covariance matrix?

- We can measure the explained variance by considering the quantity

$$\frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

- Example: