Dimensinality Reduction

Main References

Ameet Talwalkar and Henry Chai, **ML with Large Datasets**, CMU

Outline

- Data Preprocessing
- Data Visualization
- Dimensionality Reduction
- PCA
- Distributed PCA

Data Pre-processing

- ETL (extract-transfer-load)
- Cleaning data
  - Missing features/labels
  - Duplicated observations
  - Formatting errors
- Understanding data
  - Summarization
  - Exploration
  - Visualization

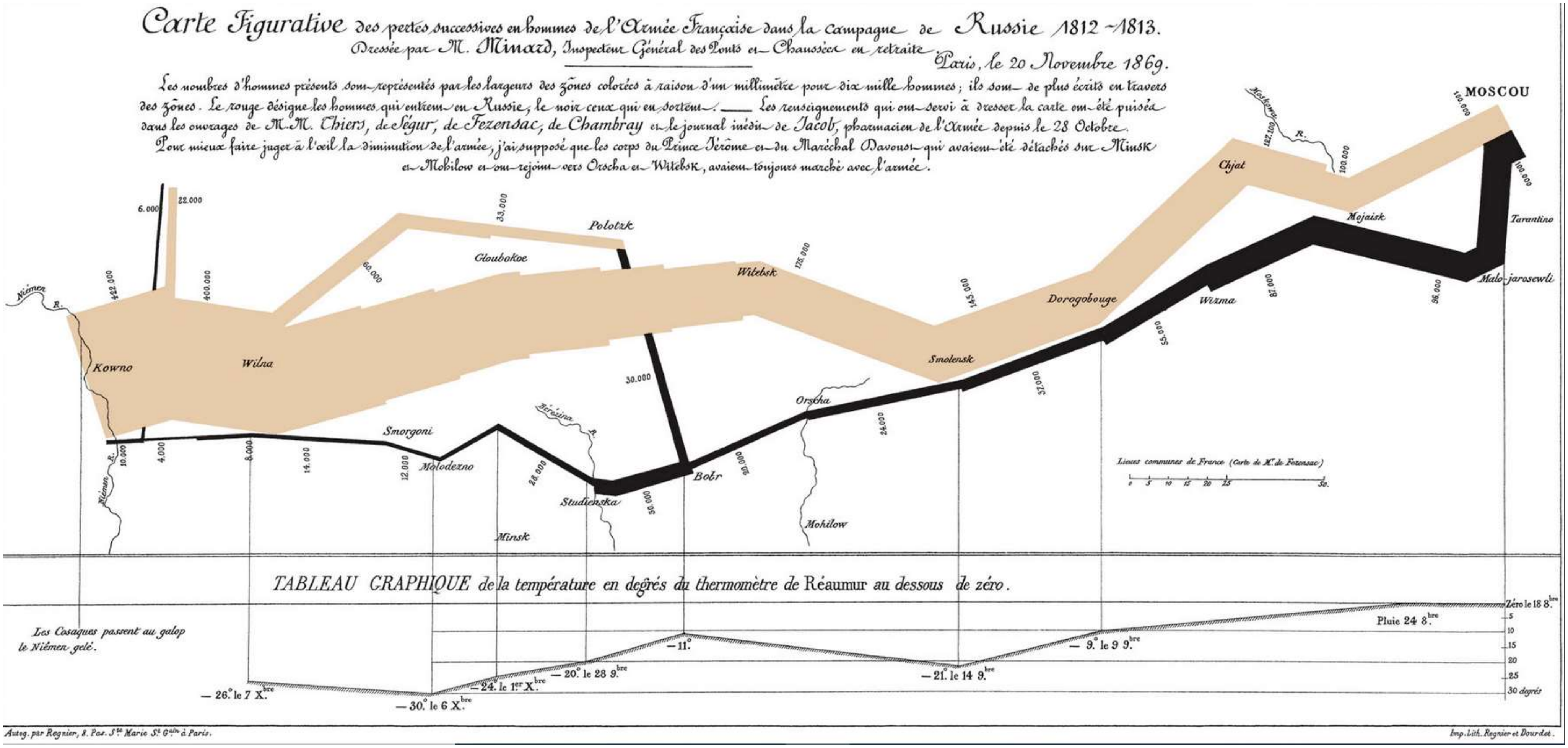
What questions can you ask better understand the data?

- Whats are features and data types?
- What are the units?
- How many observations?
- How data is collected?
- Are features are correlated?
- …

Data Visualization

- Visualizations can be used to
  - Provide insight about trends/groups/relationships
  - Reveal systematic errors
  - Aid in model selection
  - Evaluate training (e.g., measure convergence)
  - Interpret/explain predictions

Data Visualization



Charles Minard's Flow Map of Napoleon's Russian Campaign of 1812

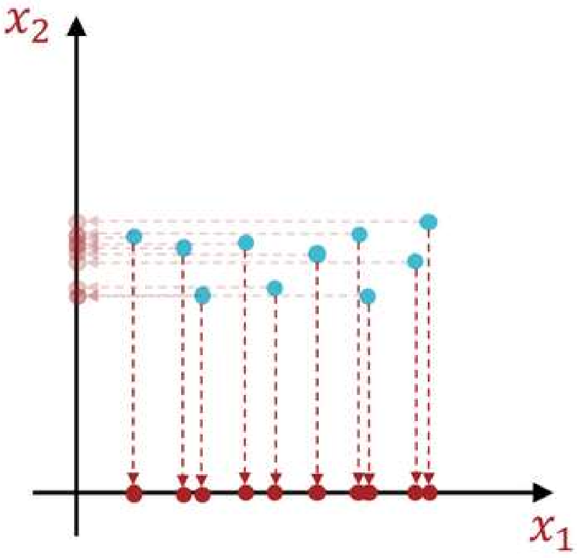
Common Data Visualizations

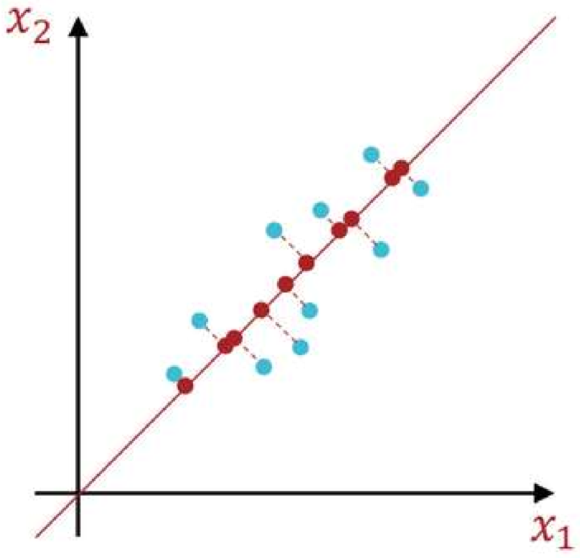
- Summary statistics
- Box plots
- Histograms
- Scatter plots

Big Data Visualizations

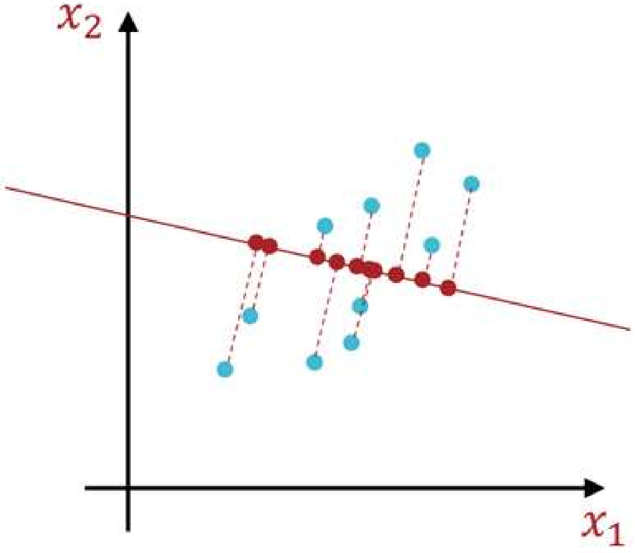
- Large n
  - Computationally expensive to render
  - Dense/complex
  - Address via subsampling or parallelization
- Large d
  - Difficult to represent more than a few dimensions
  - Address via dimensionality reduction = learning a latent (typically lower-dimensional) representation

Feature Elimination

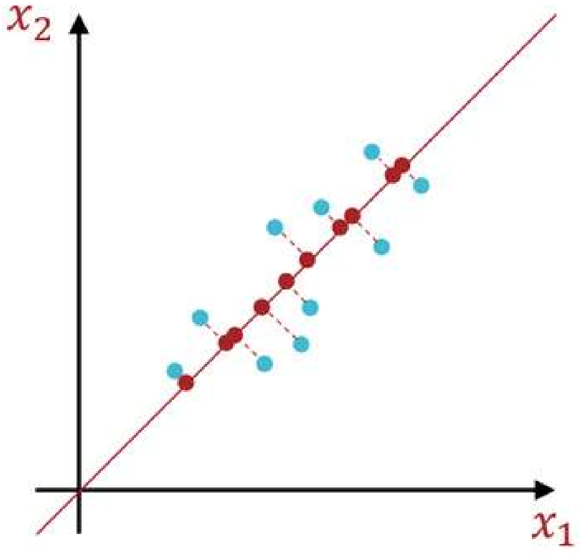




Dimensionality Reduction



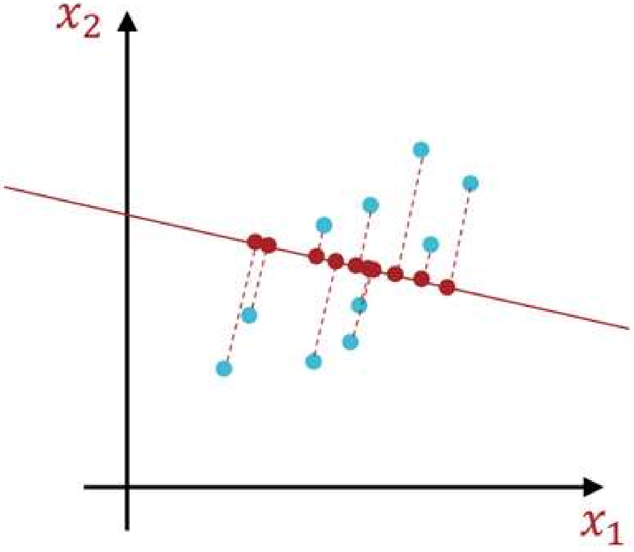
Option A



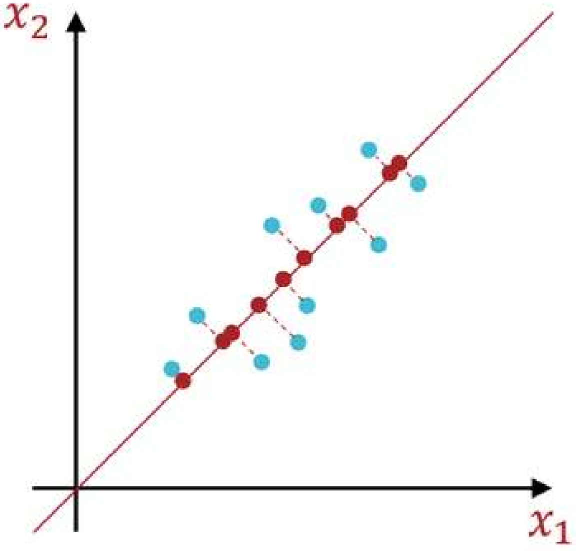
Option B

Which projection do you prefer?

Goal: minimize the reconstruction error

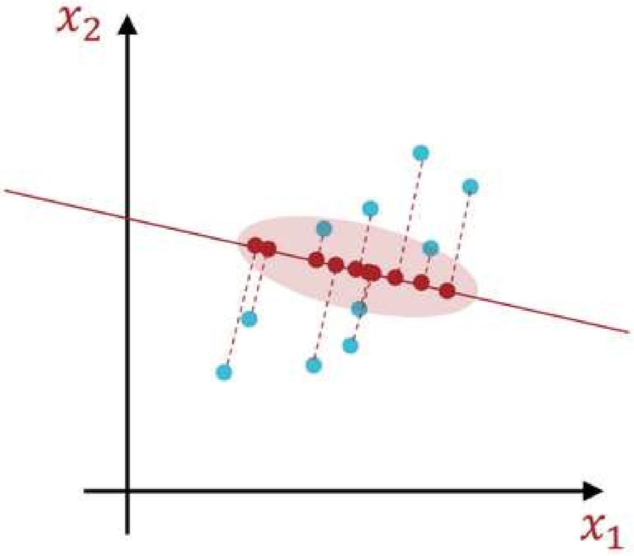


Option A

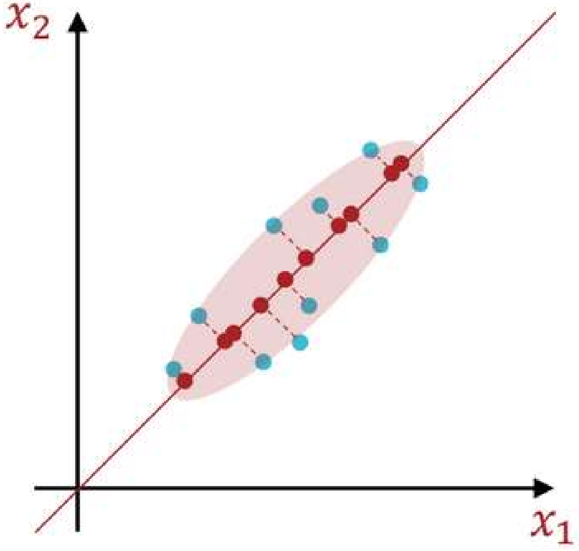


Option B

Goal: maximize the variance of the projections



Option A



Option B

Centering the Data

- To be consistent, we will constrain principal components to be orthonormal vectors (orthogonal unit vectors) that begin at the origin
- Preprocess data to be centered around the origin:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x^{(i)}$$

$$\tilde{x}^{(i)} = x^{(i)} - \mu$$

$$X = \begin{bmatrix} \tilde{x}^{(1)^T} \\ \vdots \\ \tilde{x}^{(n)^T} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

Reconstruction Error

- The projection of $\tilde{x}^{(i)}$ onto a vector v is

$$z^{(i)} = \underbrace{\left(\frac{v^T\tilde{x}^{(i)}}{\|v\|_2}\right)}_{\text{Length of projection}} \underbrace{\frac{v}{\|v\|_2}}_{\text{Direction of projection}}$$

Reconstruction Error

- The projection of $\tilde{x}^{(i)}$ onto a unit vector v is

$$z^{(i)} = \left(v^T \tilde{x}^{(i)}\right) v$$

$$\hat{v} = \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} (the\ resstruction\ error) = \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} \sum_{i=1}^{n} \left\| \tilde{x}^{(i)} - z^{(i)} \right\|_2^2$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} \sum_{i=1}^{n} \left\| \tilde{x}^{(i)} - \left(v^T \tilde{x}^{(i)}\right) v \right\|_2^2$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} \sum_{i=1}^{n} \left\| \tilde{x}^{(i)} \right\|_2^2 - \left(v^T \tilde{x}^{(i)}\right)^2$$

Minimizing the Reconstruction Error

$$\hat{v} = \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} \sum_{i=1}^{n} \left\|\tilde{x}^{(i)}\right\|_2^2 - \left(v^T \tilde{x}^{(i)}\right)^2 = \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} - \sum_{i=1}^{n} \left(v^T \tilde{x}^{(i)}\right)^2 = \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} \sum_{i=1}^{n} \left(v^T \tilde{x}^{(i)}\right)^2$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} \sum_{i=1}^{n} v^T \tilde{x}^{(i)} \tilde{x}^{(i)^T} v = \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} v^T \left(\sum_{i=1}^{n} \tilde{x}^{(i)} \tilde{x}^{(i)^T} \right) v$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} v^T (X^T X) v = \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} v^T C_X \, v$$

C_X: covariance matrix

Maximizing the Variance

$$\hat{v} = \underset{v:\|v\|_2^2=1}{\operatorname{argmax}}\, v^T C_X v$$

$$L(v, \lambda) = v^T C_X v + \lambda(\|v\|_2^2 - 1) = v^T C_X v + \lambda(v^T v - 1)$$

$$\frac{\partial L}{\partial v} = C_X v - \lambda v = 0$$

$\Rightarrow \lambda$ is eigenvalue of C_X and its corresponding eigenvector is v

Maximizing the Variance

$$\hat{v} = \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} v^T C_X v$$

$$C_X v = \lambda v \Rightarrow v^T C_X v = \lambda v^T v = \lambda$$

- The first principal component is the eigenvector $\hat{v}_1$ that corresponds to the largest eigenvalue λ_1
- The second principal component is the eigenvector $\hat{v}_2$ that corresponds to the second largest eigenvalue λ_2
- …
- λ_i is a measure of how much variance falls along $\hat{v}_i$

PCA Algorithm

- Input: $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}, k$

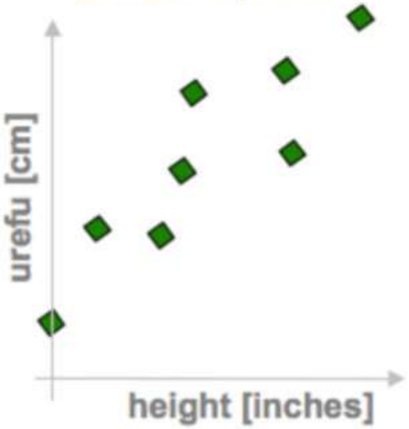
1. Center the data
   - Optionally, normalize the data by features so that all features are of the same scale
2. Compute the covariance matrix $C_X = X^T X$
3. Collect the top k eigenvectors (corresponding to the largest eigenvalues), $P \in \mathbb{R}^{n \times k}$
4. Project the data into the space defined by P, $Z = XP$

- Output: Z, the latent representation (PCA scores)

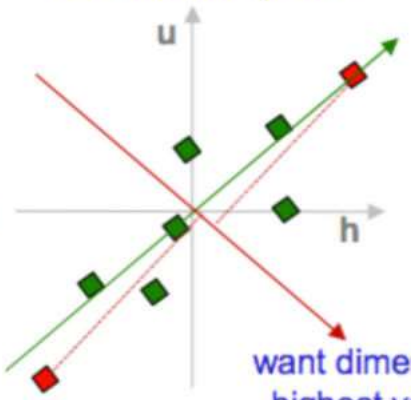
PCA in a nutshell

1. correlated hi-d data

("urefu" means "height" in Swahili)



2. center the points



want dimension of highest variance

3. compute covariance matrix

$$\begin{array}{c|cc} & h & u \\ \hline h & 2.0 & \boxed{0.8} \\ u & 0.8 & 0.6 \end{array} \rightarrow \mathrm{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$$

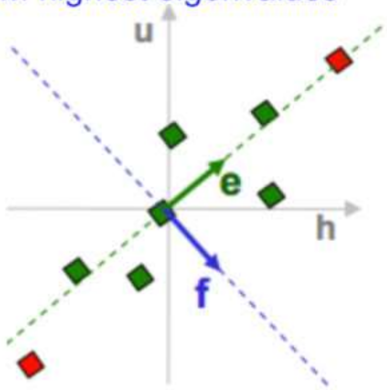
4. eigenvectors + eigenvalues

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{bmatrix} e_h \\ e_u \end{bmatrix} = \lambda_e \begin{bmatrix} e_h \\ e_u \end{bmatrix}$$

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{bmatrix} f_h \\ f_u \end{bmatrix} = \lambda_f \begin{bmatrix} f_h \\ f_u \end{bmatrix}$$

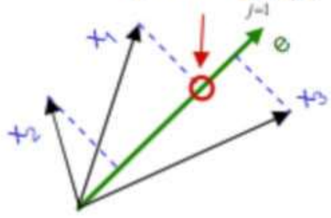
`eig(cov(data))`

5. pick m<d eigenvectors w. highest eigenvalues

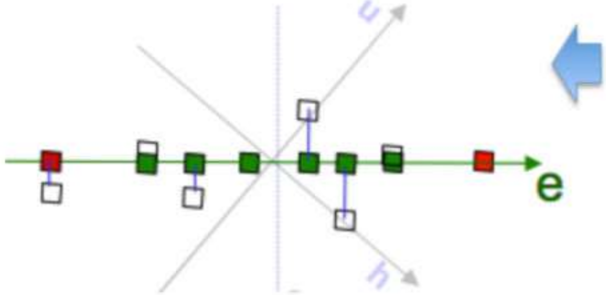


6. project data points to those eigenvectors

$$x'_e = x^T e = \sum_{j=1}^{d} x_{ij} e_j$$



7. uncorrelated low-d data



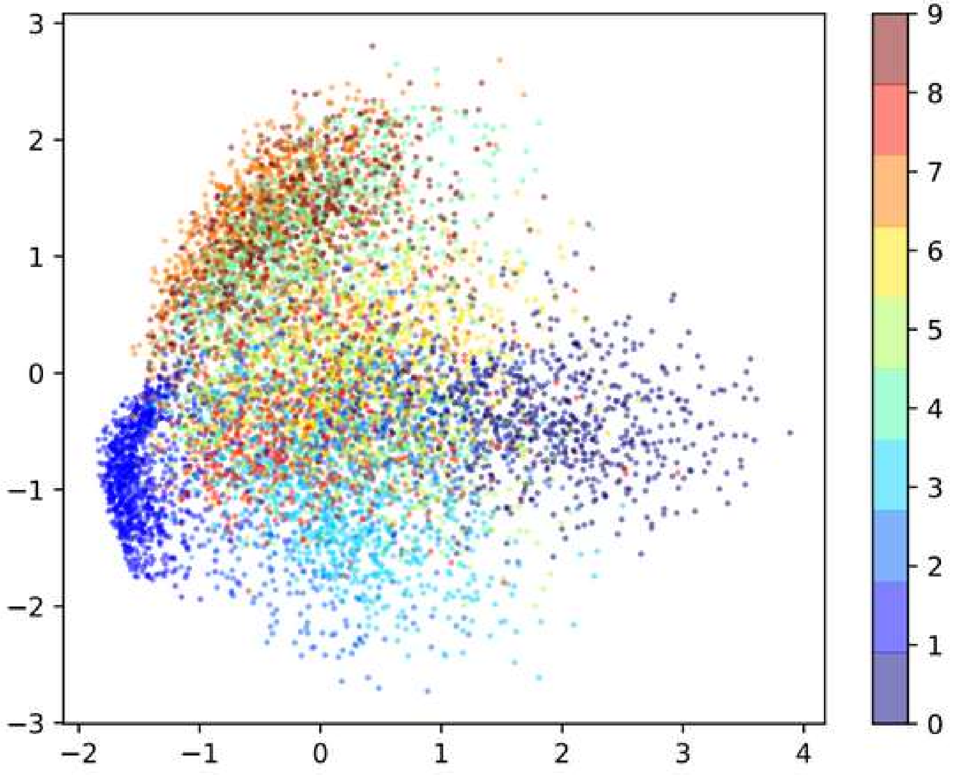
Choosing the number of PCs

- Define a percentage of explained variance for the ith PC:

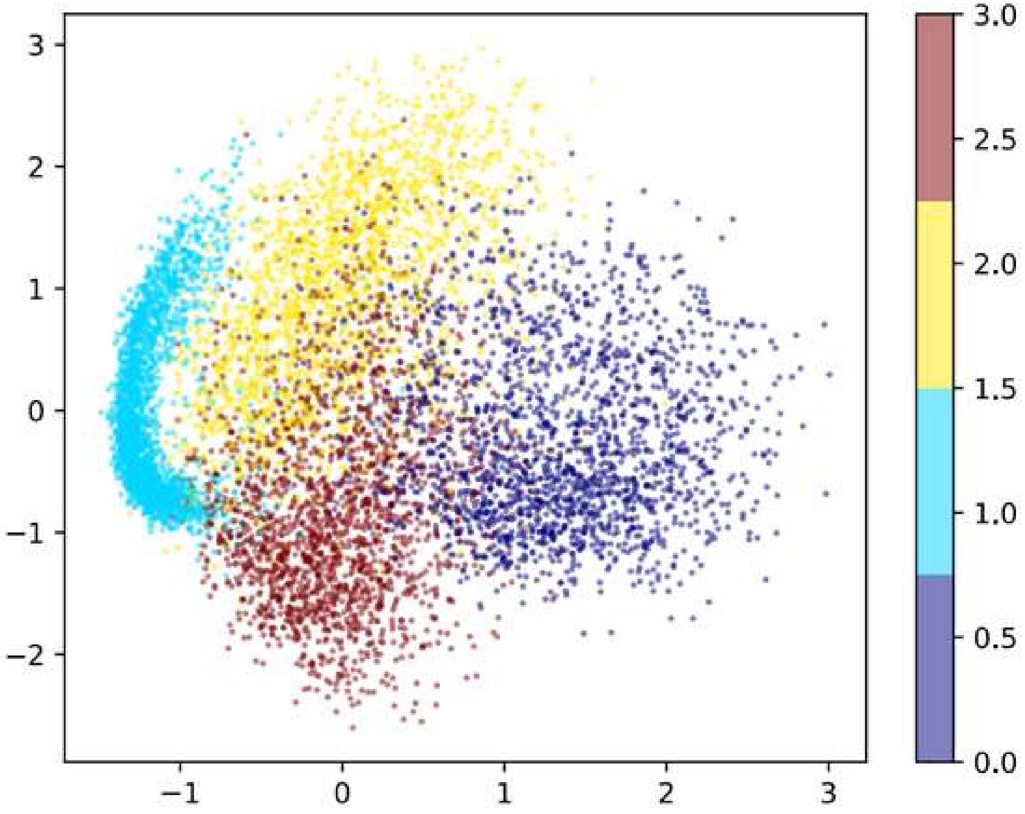
$$\frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

- Select all PCs above some threshold of explained variance, e.g., 5%
- Keep selecting PCs until the total explained variance exceeds some threshold, e.g., 90%
- Evaluate on some downstream metric

PCA Example: MNIST Digits

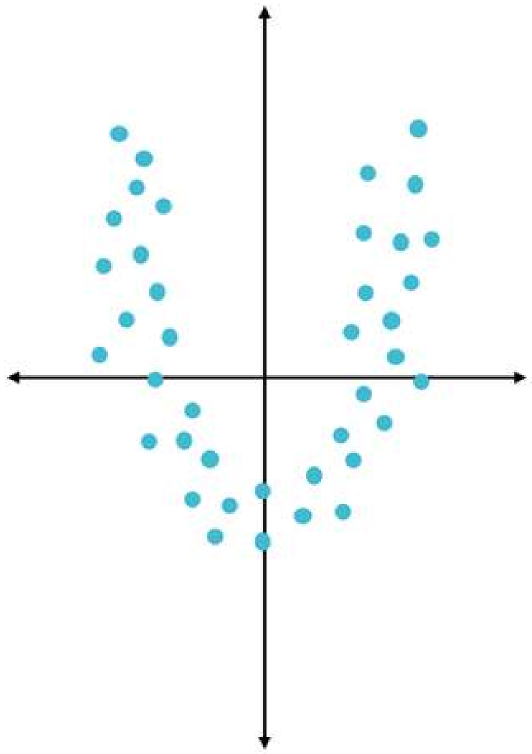


PCA Example: MNIST Digits



Shortcomings of PCA

- Sometime we don’t care about variance
- PCA only find linear combination of our features
- Principal components are expensive to compute
- Interpertability
- Principal components are orthonormal



PCA Algorithm: Computational Cost

- Input: $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}, k$

1. Center the data
   - Optionally, normalize the data by features so that all features are of the same scale
2. Compute the covariance matrix $C_X = X^T X \quad \rightarrow O(nd^2)$
3. Collect the top k eigenvectors (corresponding to the largest eigenvalues), $P \in \mathbb{R}^{n \times k} \quad \rightarrow O(d^3)$
4. Project the data into the space defined by P, $Z = XP \quad \rightarrow O(ndk)$

- Output: Z, the latent representation (PCA scores)

Nonlinear Dimensionality Reduction

- Autoencoders (1987)
- Kernel PCA (1999)
- Locally linear embedding (2000)
- Isomap (2000)
- Laplacian Eigenmaps (2003)
- t-SNE (2008)
- ... many others

PCA: Large n, Small d

- Assume $O(d^3)$ computation and $O(d^2)$ storage is possible on a single machine
  - We can store and compute the eigenvalues of $X^T X$
  - We cannot compute$X^T X$
  - We cannot store X
- Approach: basically the same as distributed linear regression
  1. Center the data in a distributed way
  2. Store the rows of X across different machines
  3. Compute $X^T X$ as the sum of outer products

Distributed Centering of the Data

Worker	$\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Reduce	$\mu = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$		
Map	$\tilde{x}^{(i)} = x^{(i)} - \mu$	$\tilde{x}^{(i)} = x^{(i)} - \mu$	$\tilde{x}^{(i)} = x^{(i)} - \mu$

μ

$O(nd)$ distributed storage (total)

$O(d)$ local work

$O(d)$ local storage

$O(nd)$ distributed work (total)

$O(nd)$ distributed storage (total)

$O(d)$ communication

Distributed Eigendecomposition of $X^T X$

$$X^T X = \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ x^{(1)} & \cdots & x^{(n)} \\ \downarrow & \downarrow & \downarrow \end{bmatrix} \begin{bmatrix} \leftarrow & x^{(1)} & \rightarrow \\ \vdots & \vdots & \vdots \\ \leftarrow & x^{(n)} & \rightarrow \end{bmatrix} = \sum_{i=1}^{n} x^{(i)} x^{(i)^T}$$

Distributed Eigendecomposition of $X^T X$

Worker	$\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\tilde{x}^{(i)} {\tilde{x}^{(i)}}^T$	$\tilde{x}^{(i)} {\tilde{x}^{(i)}}^T$	$\tilde{x}^{(i)} {\tilde{x}^{(i)}}^T$
Reduce	$eigh\left(\sum_{i=1}^{n} \tilde{x}^{(i)} {\tilde{x}^{(i)}}^T\right)$		

$O(nd)$ distributed storage (total)

$O(nd^2)$ distributed work (total) $O(d^2)$ local storage

$O(d^3)$ local work $O(d^2)$ local storage

$O(dk)$ communication

Distributed Computation of PCA Scores

Worker	$\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$O(nd)$ distributed storage (total)	
Map	$P\tilde{x}^{(i)}$	$P\tilde{x}^{(i)}$	$P\tilde{x}^{(i)}$	$O(ndk)$ distributed work (total)	$O(nk)$ local storage

PCA: Large n, Large k

- Now, $O(d^3)$ computation and $O(d^2)$ storage is not possible on a single machine
  - We cannot store and compute the eigenvalues of $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X
- Idea: use a different algorithm!
  - Turn to an iterative method for computing eigenvectors

PCA: Large n, Large k

- Now, $O(d^3)$ computation and $O(d^2)$ storage is not possible on a single machine
  - We cannot store and compute the eigenvalues of $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X
- Idea: use a different algorithm!
  - Turn to an iterative method for computing ~~eigenvectors~~ the eigenvector associated with the largest eigenvalue ($k = 1$) → power iteration

Power Iteration

- Fact: $A = X^T X$ is “diagonalizable”, i.e., any d-dimensional vector can be written as a linear combination of A’s eigenvectors:

$$b = c_1 v_1 + c_2 v_2 + \cdots + c_d v_d$$

- This follows because $X^T X$ is real and symmetric

- Assume $A = X^T X$ has one eigenvalue that is strictly larger than the others:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

Power Iteration

- Input: $A = X^T X$
- Initialize $b^{(0)}$ randomly and set $t = 0$
- While not converged
    - Update the vector b:

$$b^{(t+1)} = \frac{Ab^{(t)}}{\|Ab^{(t)}\|_2}$$

    - Increment t: $t = t + 1$
- Output: $b^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A

Power Iteration

$$b^{(0)} = c_1 v_1 + c_2 v_2 + \cdots + c_d v_d$$

$$Ab^{(0)} = c_1 A v_1 + c_2 A v_2 + \cdots + c_d A v_d$$

$$= c_1 \lambda_1 v_1 + c_2 \lambda_2 v_2 + \cdots + c_d \lambda_d v_d$$

$$A\left(Ab^{(0)}\right) = c_1 \lambda_1 A v_1 + c_2 \lambda_2 A v_2 + \cdots + c_d \lambda_d A v_d$$

$$= c_1 \lambda_1^2 v_1 + c_2 \lambda_2^2 A v_2 + \cdots + c_d \lambda_d^2 A v_d$$

$$A^t b^{(0)} = c_1 \lambda_1^t v_1 + c_2 \lambda_2^t v_2 + \cdots + c_d \lambda_d^t v_d$$

$$= \lambda_1^t \left(c_1 v_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^t v_2 + \cdots + c_d \left(\frac{\lambda_d}{\lambda_1}\right)^t v_d \right) \xrightarrow{t \to \infty} \lambda_1^t c_1 v_1$$

Power Iteration

- Input: $A = X^T X$
- Initialize $b^{(0)}$ randomly and set $t = 0$
- While not converged
  - Update the vector b:

$$b^{(t+1)} = \frac{X^T X b^{(t)}}{\|X^T X b^{(t)}\|_2}$$

  - Increment t: $t = t + 1$
- Output: $b^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A

Power Iteration

- Input: $A = X^T X$
- Initialize $b^{(0)}$ randomly and set $t = 0$
- While not converged
  - Update the vector b:

$$b^{(t+1)} = X^T X b^{(t)} = \left(\sum_{i=1}^{n} x^{(i)} {x^{(i)}}^T\right) b^{(t)}$$

$$b^{(t+1)} = \frac{b^{(t+1)}}{\|b^{(t+1)}\|_2}$$

  - Increment t: $t = t + 1$
- Output: $b^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A

Power Iteration

- Input: $A = X^T X$
- Initialize $b^{(0)}$ randomly and set $t = 0$
- While not converged
  - Update the vector b:

$$b^{(t+1)} = X^T X b^{(t)} = \sum_{i=1}^{n} x^{(i)} \left(x^{(i)^T} b^{(t)} \right)$$

$$b^{(t+1)} = \frac{b^{(t+1)}}{\|b^{(t+1)}\|_2}$$

  - Increment t: $t = t + 1$
- Output: $b^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A

Power Iteration

- Input: $A = X^T X$
- Initialize $b^{(0)}$ randomly and set $t = 0$
- While not converged
  - Update the vector b:

$$b^{(t+1)} = X^T X b^{(t)} = \sum_{i=1}^{n} x^{(i)} \left(\beta_i^{(t)}\right)$$

$$b^{(t+1)} = \frac{b^{(t+1)}}{\|b^{(t+1)}\|_2}$$

  - Increment t: $t = t + 1$
- Output: $b^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A

Distributed Power Iteration

$b^{(t)}$

Worker	$\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$O(nd)$ distributed storage (total)	
Map	$\beta_i^{(t)} = {\tilde{x}^{(i)}}^T b^{(t)}$	$\beta_i^{(t)} = {\tilde{x}^{(i)}}^T b^{(t)}$	$\beta_i^{(t)} = {\tilde{x}^{(i)}}^T b^{(t)}$	$O(nd)$ distributed work (total)	$O(n)$ distributed storage
Map	$\tilde{x}^{(i)} \beta_i^{(t)}$	$\tilde{x}^{(i)} \beta_i^{(t)}$	$\tilde{x}^{(i)} \beta_i^{(t)}$	$O(nd)$ distributed work (total)	$O(d)$ local storage
Reduce	$b^{(t+1)} = \sum_{i=1}^{n} \tilde{x}^{(i)} \beta_i^{(t)} \Big/ \left\lVert \sum_{i=1}^{n} \tilde{x}^{(i)} \beta_i^{(t)} \right\rVert_2$			$O(d)$ local work	$O(d)$ local storage

$O(d)$ communication

To find more than one eigenvector, we need to use similar (but slightly more complicated methods), e.g., PySpark's MLlib uses Krylov subspace methods

Summay

- Distributed PCA very similar to distributed linear regression
  - If d is small, simply distribute the storage and computation of $X^T X$
  - If d is large, use iterative methods (e.g., power iteration) to compute eigenvector-eigenvalue pairs