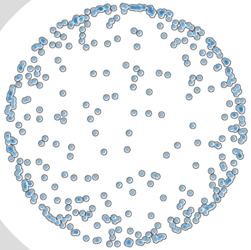
WiSe 2024/25

Machine Learning for Data Science

Lecture by G. Montavon



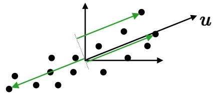
Lecture 3b | Principal Component Analysis (cont.)

Recap PCA

Two formulations of PCA:

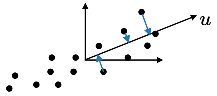
Dispersion Maximization:

- Find a projection $z = \boldsymbol{u}^\top \boldsymbol{x}$ of the data under which the dispersion (variance) is maximized.



Error Minimization:

- Find the direction that minimizes the reconstruction error (MSE) between the original data point $\boldsymbol{x}$ and its backprojection $\boldsymbol{x}^{(\text{rec})} = \boldsymbol{u}\boldsymbol{u}^\top \boldsymbol{x}$.



The two views coincide **(Pearson 1901)**

Remarks:

- PCA can be formulated as a constrained optimization problem.
- The solution of this problem is given by the eigenvector associated to the highest eigenvalue of the data covariance matrix Σ. This eigenvector can be computed in closed form using an eigenvalue solver.

Part 1

Extracting Multiple PCA Components

Extracting Multiple PCA Components

Observation:

- The basic PCA method can be seen as rewriting the data as the sum of two components, namely, what PCA is able to reconstruct, and a residue containing all what PCA cannot capture:

$$\boldsymbol{x} = \underbrace{\boldsymbol{u}\boldsymbol{u}^\top \boldsymbol{x}}_{\text{PCA}} + \underbrace{(\boldsymbol{x} - \boldsymbol{u}\boldsymbol{u}^\top \boldsymbol{x})}_{\text{Residue}}$$

- *Question:* Are there secondary principal components in the part of the data that PCA cannot capture?

Algorithm: Finding Multiple PCA Components

```
$X_{\text{res}} \leftarrow X$
for $j = 1$ to $h$ do
    $\boldsymbol{w}_j \leftarrow \text{PCA}(X_{\text{res}})$                         (i)
    $X_{\text{res}} \leftarrow X_{\text{res}} - \boldsymbol{w}_j \boldsymbol{w}_j^\top X_{\text{res}}$    (ii)
end for
```

- As an output, we get a collection of directions $\boldsymbol{w}_1, \dots, \boldsymbol{w}_h$, which we call the principal component**s**.

Extracting Multiple PCA Components

Recall:

- The principal component of a dataset corresponds to the leading eigenvector $\boldsymbol{u}_1$ of the data covariance matrix Σ.

Theoretical result:

- Assume we have extracted the sequence of principal components $(\boldsymbol{w}_1, \dots, \boldsymbol{w}_h)$ using the algorithm in the previous slide.
- One can show that these principal components are actually equivalent to the eigenvectors $(\boldsymbol{u}_1, \dots, \boldsymbol{u}_h)$ of the covariance matrix Σ, sorted by *decreasing* associated eigenvalues, i.e. $\lambda_1 > \lambda_2 > \dots > \lambda_h$.

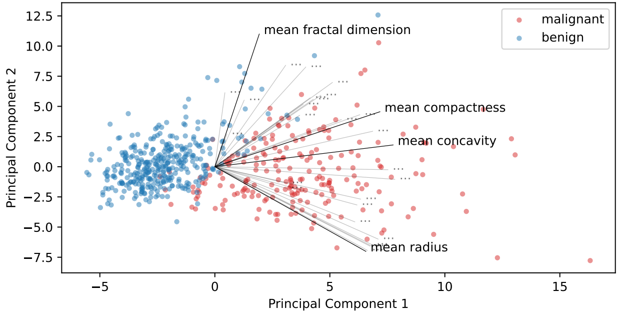
Implication:

- In practice, we do not need an iterative procedure to compute all principal components. We can instead directly compute all eigenvectors/eigenvalues of Σ (e.g. via `numpy.linalg.eigh`), and sorting them by decreasing eigenvalues.

! Full PCA solution is the collection of all eigenvectors of Σ

PCA Biplot

- Common visualization based on the *two* leading principal components. Each instance corresponds to a point in the scatter plot, and its coordinate is given by the pair $(\boldsymbol{u}_1^\top \boldsymbol{x}, \boldsymbol{u}_2^\top \boldsymbol{x})$.
- Input features can also be visualized in this plot by projecting their associate canonical coordinate vector (e.g. for feature 2, $\boldsymbol{x} = (0, 1, 0, \dots, 0)$) in PCA space. These “loading vectors” are usually depicted as arrows, shown with the feature name, and typically rescaled for visualization purposes.



Part 2

PCA and Explained Variance

PCA and Explained Variance

Recall:

- The dispersion of a multivariate dataset can be measured as the generalized variance and latter can also be expressed in terms of Σ:

$$s_{\text{tot}} = \mathbb{E}[\|\boldsymbol{x} - \boldsymbol{m}\|^2] = \sum_{j=1}^{d} \underbrace{\mathbb{E}[(x_{ij} - m_j)^2]}_{\Sigma_{jj}} = \mathrm{Tr}(\Sigma)$$

  where $\mathbb{E}[\cdot]$ denotes an average over points in the dataset.

Observation:

- The variation of the data projected on the kth principal component can be expressed as:

$$\begin{aligned} s_k &= \mathbb{E}[(\boldsymbol{u}_k^\top (\boldsymbol{x} - \boldsymbol{m}))^2] \\ &= \boldsymbol{u}_k^\top \Sigma \boldsymbol{u}_k = \lambda_k \end{aligned}$$

  i.e. it correspond to the kth eigenvalue of the covariance matrix.

PCA and Explained Variance

An Important Linear Algebra Formula:

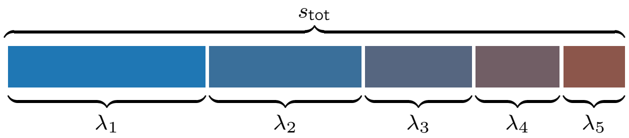
- The trace of a covariance matrix is equivalent to the sum of its eigenvalues:

$$Tr(\Sigma) = \sum_{k=1}^{h} \lambda_k$$

Implication:

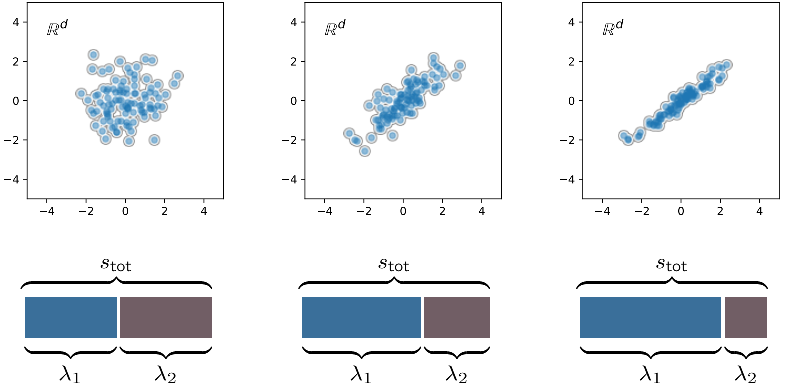
- Recalling that $s_{\text{tot}} = Tr(\Sigma)$ is the total data dispersion and λ_k is the data dispersion of the kth component, we observe that PCA produces an additive decomposition of the total dispersion in terms of principal components.

  Example for 5-dimensional data:

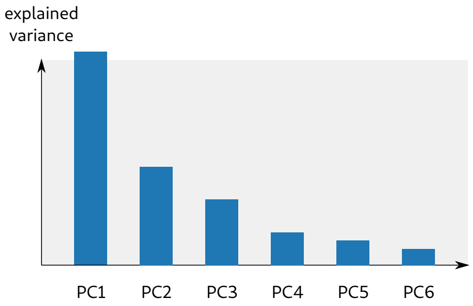


PCA and Explained Variance

Example: Three 2d datasets that have the same overall dispersion s_{tot} but distributed differently over principal components.



PCA Scree Plot

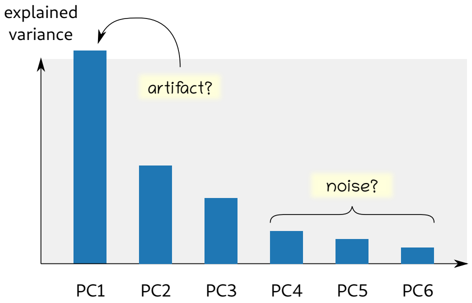


- Each bar corresponds to the explained variance associated to a particular principal component. The height of a bar is given by the associated eigenvalue. It can be interpreted as the share of the total variance explained by this component.
- The scree plot is useful to get a picture of the effective dimensionality of the data. If only the first few bars are large, it means the effective dimensionality is small and the data is therefore ‘simple’.
- Sometimes, the information contained in the scree plot is better depicted as a cumulative plot, where the kth bar then indicates the variance obtained by retaining the leading k principal components.

Part 3

PCA Beyond Describing Data

PCA Beyond Describing Data

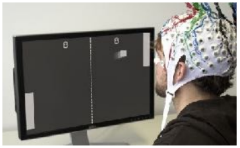
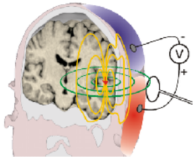
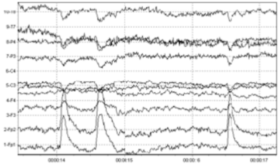


Idea:

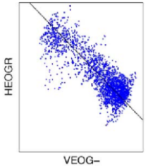
- Selectively remove certain factors that contribute to data dispersion in order to achieve desirable properties in practical applications (e.g. data denoising, artifact removal).

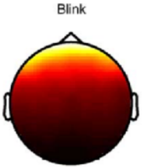
Applications of PCA: Artifact Removal

In electroencephalographic (EEG) recordings, eye blink artifacts can be stronger than the neuronal activity.

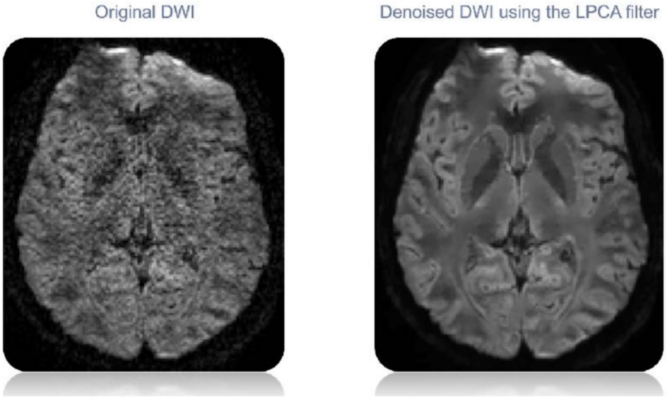



BCI2000

→ reasonable to remove first principal components





Applications of PCA: Denoising

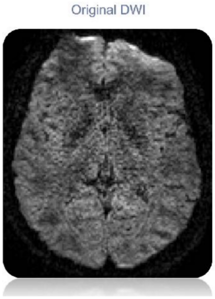


Mann et al., 2013

Part 4

Improved Computation Procedures

Motivations: $d \gg N$



Mann et al., 2013

- Many data types are high-dimensional (e.g. images or biological samples have tens of thousands of pixels/input features), i.e. $d \geq 10^4$.
- The standard implementation of PCA requires to compute a covariance matrix Σ of dimensions $d \times d$ ($\rightarrow$ large to store in memory, and most critically, *very costly eigenvalue problem*).
- Can we bypass the computation of the covariance Σ and its eigendecomposition?

PCA Computation via SVD

Singular Value Decomposition (SVD)

A singular value decomposition factorizes a matrix M as

$$M = U\Lambda V$$

where

- U contains the eigenvectors of $MM^\top$
- V contains the eigenvectors of $M^\top M$
- Λ (and Λ^2) is a diagonal matrix, with diagonal elements of Λ^2 containing the eigenvalues of $MM^\top$.

$\Rightarrow$ SVD extracts the eigenvectors and eigenvalues of the some matrix MM^T.

Reminder:

- PCA solution corresponds to the eigenvectors/eigenvalues of the covariance matrix $\Sigma = \frac{1}{N}XX^\top$.

Idea:

- We can compute PCA via SVD if we set $\Sigma = MM^\top$, in other words, by defining $M = \frac{1}{\sqrt{N}}X$.

PCA Computation via SVD

Algorithm: PCA via SVD

1. Let X be our data matrix of size $d \times N$.
2. Define $M = \frac{1}{\sqrt{N}} X$
3. Feed the matrix M to SVD and get as a result the matrices U, Λ, V.
4. PCA eigenvectors are the columns of the matrix U.
5. PCA eigenvalues are the diagonal elements of the matrix Λ^2.

Two variants of SVD (case $d > N$):

1. `full_matrices=True`: Include all eigenvectors/eigenvalues of the matrix $MM^\top$.

$$\overbrace{M}^{d \times N} = \overbrace{U}^{d \times d} \overbrace{\Lambda}^{d \times d} \overbrace{V}^{d \times N}$$

2. `full_matrices=False`: Observe that M (and $MM^\top$) has rank N and that there are therefore only N eigenvectors of the matrix $MM^\top$ with non-zero eigenvalues. Only retain those eigenvectors.

$$\overbrace{M}^{d \times N} = \overbrace{U}^{d \times N} \overbrace{\Lambda}^{N \times N} \overbrace{V}^{N \times N}$$

PCA Computation via SVD

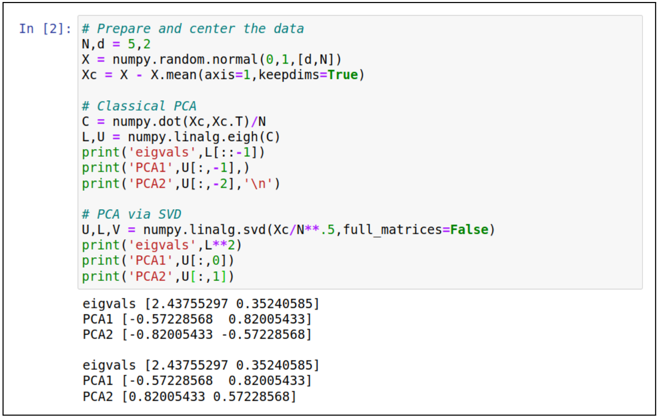
$$\overbrace{M}^{d\times N} = \overbrace{U}^{d\times N} \overbrace{\Lambda}^{N\times N} \overbrace{V}^{N\times N}$$

What did we gain? (case $d > N$)

- No need to compute the matrix Σ of size $d \times d$. (The matrix $MM^\top$ of which SVD extracts the eigenvectors/values has size $d \times d$ but it is never computed/stored explicitly.)
- SVD of a matrix M of size $d \times N$ is much faster that computing the full eigendecomposition of a matrix Σ of size $d \times d$.
- Therefore, we can compute PCA for very high-dimensional data, as long as the number of examples N remains not too large.

PCA Computation via SVD

- Verification that 'Classical PCA' and 'SVD PCA' give the same output



```
In [2]: # Prepare and center the data
        N,d = 5,2
        X = numpy.random.normal(0,1,[d,N])
        Xc = X - X.mean(axis=1,keepdims=True)

        # Classical PCA
        C = numpy.dot(Xc,Xc.T)/N
        L,U = numpy.linalg.eigh(C)
        print('eigvals',L[::-1])
        print('PCA1',U[:,-1],)
        print('PCA2',U[:,-2],'\n')

        # PCA via SVD
        U,L,V = numpy.linalg.svd(Xc/N**.5,full_matrices=False)
        print('eigvals',L**2)
        print('PCA1',U[:,0])
        print('PCA2',U[:,1])
```

```
eigvals [2.43755297 0.35240585]
PCA1 [-0.57228568  0.82005433]
PCA2 [-0.82005433 -0.57228568]

eigvals [2.43755297 0.35240585]
PCA1 [-0.57228568  0.82005433]
PCA2 [0.82005433 0.57228568]
```

- Eigenvalues are the **same**.
- PCA projection vector is the **same** $\underbrace{\text{up to a possible sign flip}}_{\substack{\text{Not a problem! PCA components} \\ \text{are defined anyways up to a sign flip.}}}$.

PCA Computation via SVD, Limitations

- The SVD has computational complexity of $\mathcal{O}(\min(N^2 d, d^2 N))$.
- When $d \approx N$, the complexity is roughly as bad as the that of computing the eigenvectors of Σ, that is, $\mathcal{O}(d^3)$.
- This makes the computation of principal components using SVD prohibitive for large datasets.

Note:

- In practice, we often need **only the first few principal components**, whereas SVD computes all of them.

Question:

- Can we design a faster procedure that only extracts the first few principal components?

Power Iteration, Algorithm

Procedure to find the first principal component.

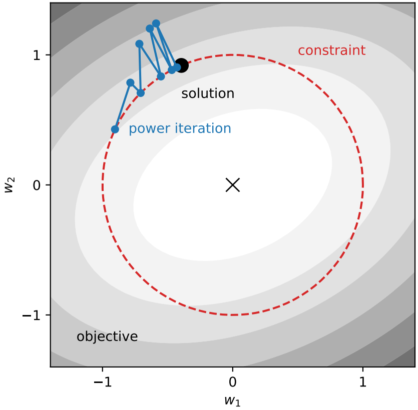
Power Iteration (POWIT)

```
$\boldsymbol{u} \sim$ random$(\dots)$
repeat
    $\boldsymbol{v} \leftarrow \Sigma \boldsymbol{u}$
    $\boldsymbol{u} \leftarrow \boldsymbol{v} / \|\boldsymbol{v}\|$
until convergence
```

Questions:

- Does it always converge? **yes**
- How fast it converges? **exponentially fast**

(Homework: prove this.)



Power Iteration (more components)

The power iteration (POWIT) method only gives us the leading eigenvector. If we want further PCA components, we need to compute them iteratively.

Naive approach: Recompute covariance at each step

```
for $j = 1$ to $h$ do
    $\Sigma = \frac{1}{N} X X^\top$
    $\boldsymbol{u}_j \leftarrow \text{POWIT}(\Sigma)$
    $X \leftarrow X - \boldsymbol{u}_j \boldsymbol{u}_j^\top X$
end for
```

Better approach: Work directly in covariance space

```
$\Sigma = \frac{1}{N} X X^\top$
for $j = 1$ to $h$ do
    $\boldsymbol{u}_j \leftarrow \text{POWIT}(\Sigma)$
    $\Sigma \leftarrow \Sigma - \boldsymbol{u}_j \boldsymbol{u}_j^\top \Sigma$
end for
```

Summary

Summary

- In practice, we are not reduced to compute a single principal component. We can compute multiple principal components, and the latter correspond to the collection of *all* eigenvectors of the data covariance matrix Σ.
- A **PCA biplot** visualizes data in terms of the two leading principal components. It provides further insights compared to a simple histogram along a single principal component.
- Each principal component explains a certain share of the total variance in the data. The contribution of each principal component is typically conveyed in a **PCA scree plot**. The latter is useful to determine the effective dimensionality of the data.
- PCA has multiple applications beyond describing and analyzing the data. This includes, in particular, **artifact reduction** and **data denoising**.
- Several methods exist to compute principal components (e.g. **SVD**, **Power Iteration**). Which one to choose is dataset-dependent and also application-dependent.