Principal Component Analysis

CS498
Today’s lecture

• Adaptive Feature Extraction

• Principal Component Analysis
  – How, why, when, which
A dual goal

• Find a good representation
  – The features part

• Reduce redundancy in the data
  – A side effect of “proper” features
Example case

- Describe this input
What about now?
A “good feature”

- “Simplify” the explanation of the input
  - Represent repeating patterns
  - When defined makes the input simpler

- How do we define these abstract qualities?
  - On to the math …
Linear features

\[ Z = WX \]
A 2D case

\[ Z = WX = \begin{bmatrix} z_1^T & w_1^T & x_1^T \\ z_2^T & w_2^T & x_2^T \end{bmatrix} \]

Matrix representation of data

Scatter plot of same data
Defining a goal

- Desirable feature features
  - Give “simple” weights
  - Avoid feature similarity

- How do we define these?
One way to proceed

- “Simple weights”
  - Minimize relation of the two dimensions

- “Feature similarity”
  - Same thing!
One way to proceed

• “Simple weights”
  – Minimize relation of the two dimensions
  – Decorrelate: $z_1^Tz_2 = 0$

• “Feature similarity”
  – Same thing!
  – Decorrelate: $w_1^Tw_2 = 0$
Diagonalizing the covariance

- Covariance matrix

\[ \text{Cov}\left( z_1, z_2 \right) = \begin{bmatrix} z_1^T z_1 & z_1^T z_2 \\ z_2^T z_1 & z_2^T z_2 \end{bmatrix} / N \]

- Diagonalizing the covariance suppresses cross-dimensional co-activity
  - if \( z_1 \) is high, \( z_2 \) won’t be, etc

\[ \text{Cov}\left( z_1, z_2 \right) = \begin{bmatrix} z_1^T z_1 & z_1^T z_2 \\ z_2^T z_1 & z_2^T z_2 \end{bmatrix} / N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \]
Problem definition

• For a given input $X$

• Find a feature matrix $W$

• So that the weights decorrelate

\[
(WX)(WX)^T = NI \Rightarrow ZZ^T = NI
\]
How do we solve this?

- Any ideas?

\[
(WX)(WX)^T = NI
\]
Solving for diagonalization

\[
(WX)(WX)^T = NI \Rightarrow
\]
\[
\Rightarrow WXX^T W^T = NI \Rightarrow
\]
\[
\Rightarrow WCov(X) W^T = I
\]
Solving for diagonalization

• Covariance matrices are positive definite
  – Therefore symmetric
    • have orthogonal eigenvectors and real eigenvalues
  – and are factorizable by:

\[ U^T A U = \Lambda \]

– Where \( U \) has eigenvectors of \( A \) in its columns
– \( \Lambda = \text{diag}(\lambda_i) \), where \( \lambda_i \) are the eigenvalues of \( A \)
Solving for diagonalization

The solution is a function of the eigenvectors $U$ and eigenvalues $\Lambda$ of $\text{Cov}(X)$

\[ W\text{Cov}(X)W^T = I \Rightarrow \]

\[ \Rightarrow W = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix}^{-1} U^T \]
So what does it do?

- **Input data covariance:** \( \text{Cov}(X) \approx \begin{bmatrix} 14.9 & 0.05 \\ 0.05 & 1.08 \end{bmatrix} \)

- **Extracted feature matrix:** \( W \approx \begin{bmatrix} 0.12 & -30.4 \\ -8.17 & -0.03 \end{bmatrix} \div N \)

- **Weights covariance:** \( \text{Cov}(WX) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \)
Another solution

- This is not the only solution to the problem
- Consider this one:

\[
(WX)(WX)^T = NI \Rightarrow \\
WXX^TW^T = NI \Rightarrow \\
W = (XX^T)^{-1/2}
\]
Another solution

• This is not the only solution to the problem

• Consider this one:

\[
(WX)(WX)^T = NI \Rightarrow \\
WXX^TW^T = NI \Rightarrow \\
W = (XX^T)^{-1/2} \Rightarrow \\
W = US^{-1/2}V^T \\
[U, S, V] = SVD\left(XX^T\right)
\]
Decorrelation in pictures

- An implicit Gaussian assumption
  - $N$-D data has $N$ directions of variance
Undoing the variance

- The decorrelating matrix $W$ contains two vectors that normalize the input’s variance.
Resulting transform

- Input gets scaled to a well behaved Gaussian with unit variance in all dimensions
A more complex case

- Having correlation between two dimensions
  - We still find the directions of maximal variance
  - But we also rotate in addition to scaling
One more detail

- So far we considered zero-mean inputs
  - The transforming operation was a rotation
- If the input mean is not zero bad things happen!
  - Make sure that your data is zero-mean!
Principal Component Analysis

- This transform is known as PCA
  - The features are the principal components
    - They are orthogonal to each other
    - And produce orthogonal (white) weights
  - Major tool in statistics
    - Removes dependencies from multivariate data

- Also known as the KLT
  - Karhunen-Loeve transform
A better way to compute PCA

• The Singular Value Decomposition way

\[ [U, S, V] = \text{SVD}(A) \Rightarrow A = USV^T \]

• Relationship to eigendecomposition
  – In our case (covariance input \( A \)), \( U \) and \( S \) will hold the eigenvectors/values of \( A \)

• Why the SVD?
  – More stable, more robust, fancy extensions
PCA through the SVD

- Feature Matrix
- √eigenvalue matrix
- Weight Matrix

= SVD

- Input Matrix
- Input Covariance

Dimensions ➔ Features ➔ Samples ➔ Samples ➔ Features ➔ Features ➔ Dimensions ➔ Dimensions

= SVD
Dimensionality reduction

• PCA is great for high dimensional data

• Allows us to perform dimensionality reduction
  – Helps us find relevant structure in data
  – Helps us throw away things that won’t matter
A simple example

• Two very correlated dimensions
  – e.g. size and weight of fruit
  – One effective variable

• PCA matrix here is:

\[ W = \begin{bmatrix}
-0.2 & -0.13 \\
-13.7 & 28.2
\end{bmatrix} \]

– Large variance between the two components
  • about two orders of magnitude
A simple example

• Second principal component needs to be super-boosted to whiten the weights
  – maybe is it useless?

• Keep only high variance
  – Throw away components with minor contributions
What is the number of dimensions?

- If the input was $M$ dimensional, how many dimensions do we keep?
  - No solid answer (estimators exist but are flaky)

- Look at the singular/eigen-values
  - They will show the variance of each component, at some point it will be small
Example

- Eigenvalues of 1200 dimensional video data
  - Little variance after component 30
  - We don’t need to keep the rest of the data
So where are the features?

• We strayed off-subject
  – What happened to the features?
  – We only mentioned that they are orthogonal

• We talked about the weights so far, let’s talk about the principal components
  – They should encapsulate structure
  – How do they look like?
Face analysis

- Analysis of a face database
  - What are good features for faces?

- Is there anything special there?
  - What will PCA give us?
  - Any extra insight?

- Lets use MATLAB to find out …
The Eigenfaces
Low-rank model

- Instead of using 780 pixel values we use the PCA weights (here 50 and 5)
PCA for large dimensionalities

- Sometimes the data is high dim
  - e.g. videos 1280x720xT = 921,600D x T frames

- You will not do an SVD that big!
  - Complexity is $O(4m^2n + 8mn^2 + 9n^3)$

- Useful approach is the EM-PCA
EM-PCA in a nutshell

• Alternate between successive approximations
  – Start with random $C$ and loop over:
    $$Z = C^+ X$$
    $$C = XZ^+$$
  – After convergence $C$ spans the PCA space

• If we choose a low rank $C$ then computations are significantly more efficient than the SVD
  – More later when we cover EM
PCA for online data

- Sometimes we have too many data samples
  - Irrespective of the dimensionality
  - e.g. long video recordings

- Incremental SVD algorithms
  - Update the $U, S, V$ matrices with only a small set or a single sample point
  - Very efficient updates
A Video Example

• The movie is a series of frames
  – Each frame is a data point
  – 126, 80x60 pixel frames
  – Data will be 4800x126

• We can do PCA on that
PCA Results

Video Component 1  Video Component 2  Video Component 3

Component weights

Time
PCA for online data II

- “Neural net” algorithms
- Naturally online approaches
  - With each new datum, PC’s are updated
- Oja’s and Sanger’s rules
  - Gradient algorithms that update $W$
- Great when you have minimal resources
PCA and the Fourier transform

- We’ve seen why sinusoids are important
  - But can we statistically justify it?

- PCA has a deep connection with the DFT
  - In fact you can derive the DFT from PCA
An example

• Let’s take a time series which is not “white”
  – Each sample is somewhat correlated with the previous one (Markov process)

\[
X = \begin{bmatrix}
x(t), & \cdots, & x(t + T)
\end{bmatrix}
\]

• We’ll make it multidimensional

\[
\begin{bmatrix}
x(t) & x(t + 1) \\
\vdots & \vdots & \ddots \\
x(t + N) & x(t + 1 + N)
\end{bmatrix}
\]
An example

• In this context, features will be repeating temporal patterns smaller than \( N \)

\[
Z = W \begin{bmatrix}
    x(t) & x(t+1) \\
    \vdots & \vdots \\
    x(t+N) & x(t+1+N)
\end{bmatrix}
\]

• If \( W \) is the Fourier matrix then we are performing a frequency analysis
PCA on the time series

• By definition there is a correlation between successive samples

\[
\text{Cov}(X) \approx \begin{bmatrix}
1 & 1-e & \cdots & 0 \\
1-e & 1 & 1-e & \vdots \\
\vdots & 1-e & 1 & 1-e \\
0 & \cdots & 1-e & 1
\end{bmatrix}
\]

• Resulting covariance matrix will be symmetric Toeplitz with a diagonal tapering towards 0
Solving for PCA

- The eigenvectors of Toeplitz matrices like this one are (approximately) sinusoids
And ditto with images

- Analysis of coherent images results in 2D sinusoids.
So now you know

• The Fourier transform is an “optimal” decomposition for time series
  – In fact you will often not do PCA and do a DFT

• There is also a loose connection with our perceptual system
  – We kind of use similar filters in our ears and eyes (but we’ll make that connection later)
Recap

• Principal Component Analysis
  – Get used to it!
  – Decorrelates multivariate data, finds useful components, reduces dimensionality

• Many ways to get to it
  – Knowing what to use with your data helps

• Interesting connection to Fourier transform
Check these out for more

• Eigenfaces

• Incremental SVD

• EM-PCA
  – http://cs.nyu.edu/~roweis/papers/empca.ps.gz