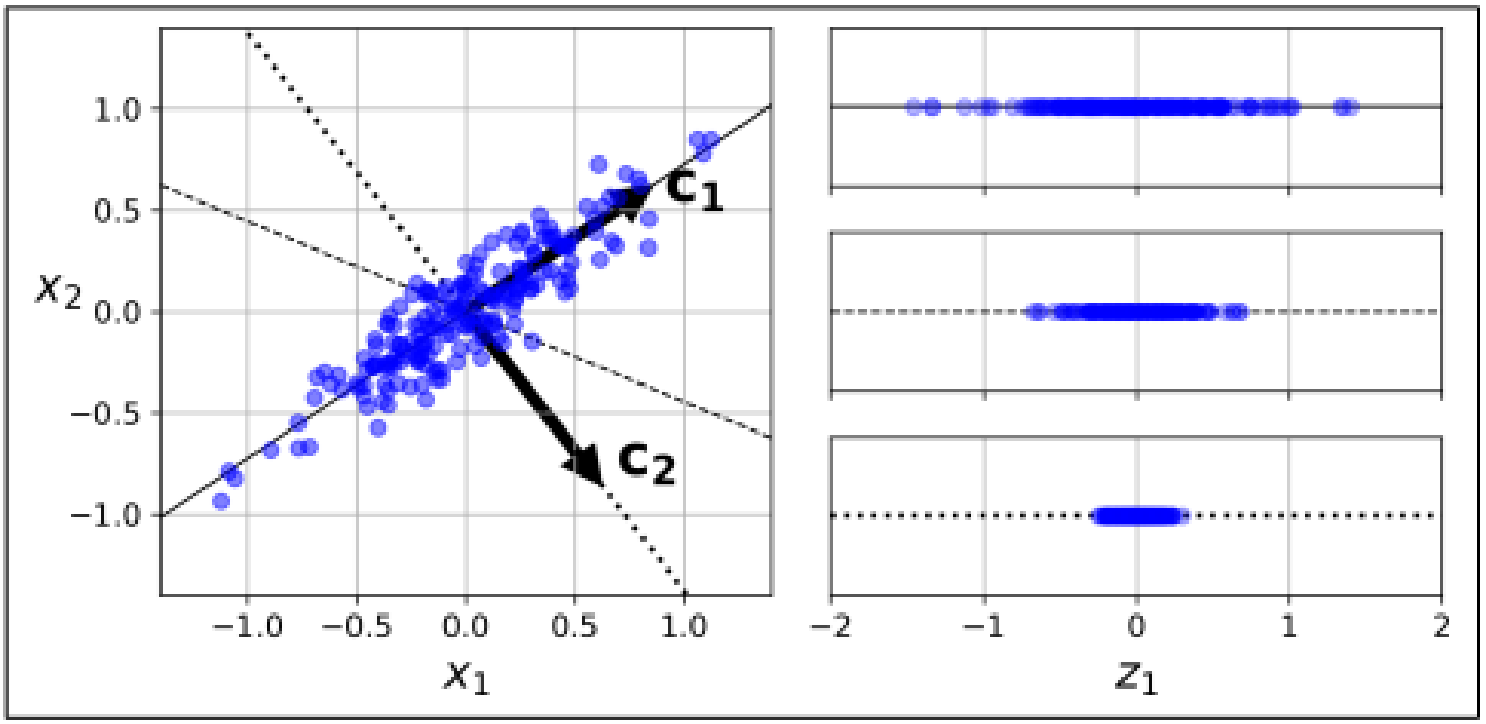


Understanding Principal Component Analysis

Table Of Contents

(*) Principal Component Analysis

→ The most popular dimensionality reduction technique is the principal component analysis, it is an unsupervised algorithm used for feature selection.

Idea behind PCA

→ PCA transforms and fits the data from a higher dimensional space to a lower dimensional subspace, this results into an entirely new co-ordinates system at the point where the first axis explains the most variance in the data (first principle component), this is the idea behind PCA.

what are principal components

→ Principal components are the axis, that explain the maximum variance in the data, the first principal component explains the most variance, the 2nd a bit less and so on..

→ Each of new axis / principal component found using PCA is a linear combination of the original features

→ All the principal component are uncorrelated and orthogonal to each other.

(*) Math Behind PCA

→ Principal component analysis can be implemented in two approaches, each with its unique focus and method. They are as follows:

① Eigen decomposition method

② singular value decomposition method.

Let's delve into them.

① Eigen Decomposition Method

→ Eigen decomposition is a process that decomposes a square matrix A into a set of eigenvectors and eigenvalues.

→ For a given square matrix A, the eigenvectors are the non-zero vectors that satisfy the equation.

$$\boxed{A = \lambda V}$$ where,

V is the eigen vector and

λ is the corresponding eigenvalue.

Steps of Eigen-decomposition approach

① Standardize columns of A, so that each feature has a mean of zero, this ensures that each feature contributes equally to the analysis.

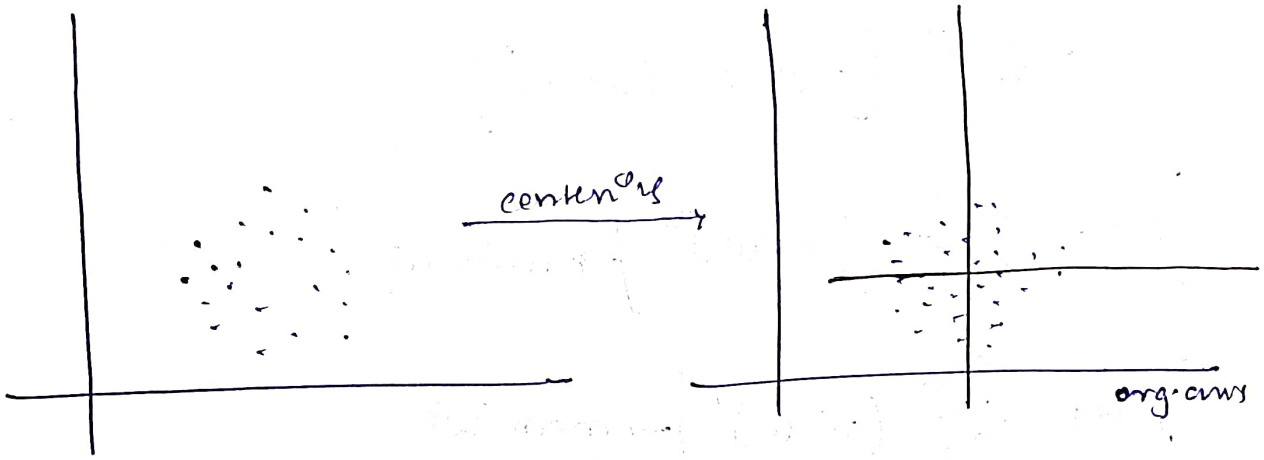
(2) compute co-variance matrix from the normalized data, $\Sigma = A^T A / (m-1)$

(3) perform eigen decomposition of Σ (covariance matrix), to obtain eigen values & eigenvectors.

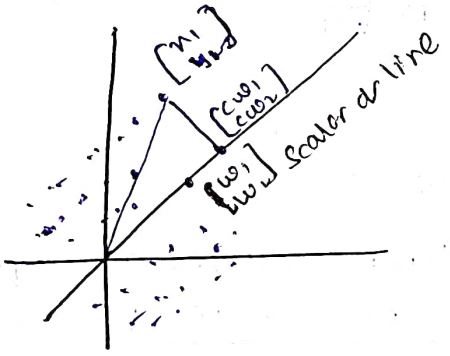
(4) choose the top K eigen vectors compat associated with the largest eigen values to construct the principal components as AX_K → Kth eigen vector.

PROOF

Step 1



we have, $D = \{x_1, x_2, \ldots x_n\}$ $x_i \in \mathbb{R}^d$



the projection point will be, $(x^T w) \cdot w$, considering w is a scalar or the best fit line/ the line that has least reconstruction error.

$$\text{Error}(\text{line}, \text{dataset}) = \sum_{i=1}^{n} \text{error}(\text{line}, x_i)$$

$$= \sum_{i=1}^{n} \| x_i - (x^T w) w \|^2$$

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} \| x - (x^T w)\cdot w \|^2$$

This is the mean of the residuals, we can think of this as the avg. error.

$$= \frac{1}{n}\sum_{i=1}^{n} \left[(x - (x^T w)\cdot w)^T \cdot (x - (x^T w)\cdot w) \right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left[x_i^T x_i - (x_i^T w)^T - (x_i^T w)^2 + [x_i^T w^2] \right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left[x_i^T x_i - (x_i^T w)^2 \right]$$

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} -(x_i^T w)^2 \rightarrow \min w$$

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} (x_i^T w)^2 \rightarrow \max w$$

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} (x_i^T w)(w^T x_i) = \frac{1}{n}\sum_{i=1}^{n} w^T (x_i \cdot x_i^T) w$$

$$P(w) = w^T \left(\frac{1}{n}\sum_{i=1}^{n} x_i \cdot x_i^T \right) \cdot w$$

we know this is covariance matrix

Each column corresponds to one data point.

$$\therefore \max(w) = w^T C \cdot w$$

where, C – covariance matrix
w – Eigen vector corresponding to max eigen value of C

→ so, Basically we can say that, the line which represents the data points or minimum reconstruction error is same as the line which maximizes the $\boxed{w^T C w}$, as C - covariance matrix,
and the line is given by eigen vector corresponding to max. eigen value of C, and this line is the principal component!!

★ $\boxed{\text{First principal component } (w_1) = \arg\max_{w} (w^T C \cdot w)}$

→ Now, any line which minimizes the sum of errors / residuals, must also be orthogonal to w_1.
Hence, second principal component will be orthogonal to the first $\boxed{w_2^T w_1 = 0}$ ★

→ By, continuing this procedure, we get $\{w_1, w_2 \dots w_d\}$
Such that,

$$\boxed{\begin{aligned} &\|w_k^2\| = 1 \quad \forall k \text{ and} \\ &w_i^T w_j = 0 \quad \forall i \neq j \end{aligned}}$$

Residuals after d rounds

$$\forall i \rightarrow x_i - \left[(x_i^T w_1) w_1 + (x_i^T w_2) w_2 + \dots + (x_i^T w_d) w_d\right] = \vec{0}$$

$$\forall i \rightarrow x_i = (x_i^T w_1) w_1 + (x_i^T w_2) w_2 + \dots + (x_i^T w_d) \cdot w_d$$

→ If original dataset is of (d×n), then based on above eqn, after k rounds of PCA, we get $\boxed{(d \times k) + (k \times n)}$

Eg

→ Suppose we have an initial dataset with 50 features and 100 samples. $d=50$, $n=100$

Data dimension: $50 \times 100 = 1500$ data points to represent entire dataset

After 5 PCA rounds, $K=5$

Data dimension $= (50 \times 5) + (5 \times 100) = 750$

So, now we only need half of datapoints to represent entire dataset

→ $\{w_1\ w_2\ w_3\ w_4\ w_5\}$ → Representation, common for dataset

→ $x_i \rightarrow [x_i^T w_1\ \ x_i^T w_2\ \ x_i^T w_3\ \ x_i^T w_4\ \ x_i^T w_5]$ → Data Point Specific

→ Hence, to reconstruct x_i we don't need 1000 numbers, like naive way features

we only need 5 principal components

→ And through Eigen decomposition approach, we calculate the covariance matrix, and find its eigenvalues & eigenvectors. And define the principal components from eigen vectors. We can reconstruct by $A \times B \times B^T$

② Singular value decomposition Method

→ SVD method factorises the dataset in such a way that it becomes a product of the multiplication of 3 individual matrices.

$$X(\text{original data}) = U * \Sigma * V^T$$

where, U is the matrix that contains the Principal components.

Steps in SVD approach

① Standardize columns of A, to ensure each feature contributes equally to analysis.

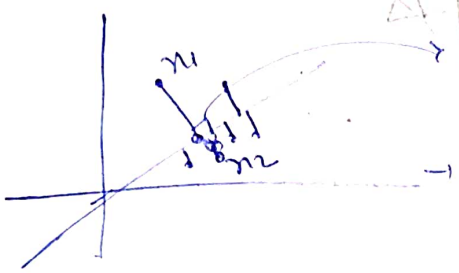
② compute SVD of the centered data matrix to obtain matrices U, Σ, and V^T, such that $X = U \Sigma V^T$ where U contains left singular vectors, Σ holds the singular values & V^T includes the right singular vectors.

③ select principal components: the Principal components are obtained by selecting the columns of U corresponding to the largest singular values in Σ, these Principal components represent the directions of maximum variance in the dataset.

Note:-

→ what does high variance mean?

when making a proxy,
what if two points get same proxy.



→ Now, How do we distinguish x_1 & x_2.

→ And How do we reconstruct x_1 & x_2.

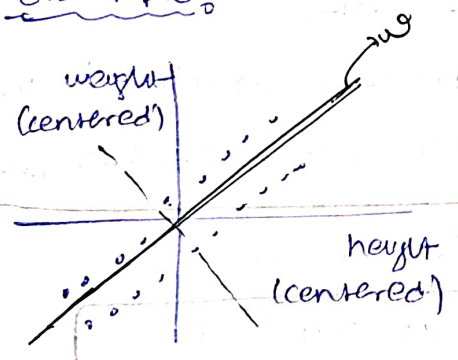
→ So, if all the datapoints are crowded, then we can't reconstruct and the information is lost.

This happens due to less variance / small variance.

Hence, we want directions where projections not crowded.

variance has to be high,
so the datapoints will be spread out
and reconstruction of points can be done properly.

Example:



- height gives some information weight

→ we have two directions now w_1, w_2..

→ And these two directions are decorrelated, every direction is giving a new information from the other.

→ Hence, we can say these directions are de-correlated.

this's what the algorithm is essentially doing & it's called Principal Component Analysis.

* Once we get principal components, Project the original data onto the subspace spanned by the selected principal components, effectively reducing the dimensionality of the dataset.

(*) What is the optimal no. of PC's needed?

* Before we ask this qn, we need to ask how much of the information in a given dataset is lost by projecting the observations onto first few principal components, i.e.,

> the proportion of variance explained by each principal component.

Explained variance ratio = $\frac{\text{variance by } m^{th} \text{ PC}}{\text{total variance}}$

$$EVR = \frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{p} w_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$$

→ EVR of PC1 = $\frac{\text{Distance of PC1 Points}}{\text{Distance of PC1 Points} + PC2 + \dots PCn}$

If EVR of PC1 = 0.91, then that means PC1 explains 91% of the variance of data.

→ Now that we have a metrics to get the variance explained, we can choose the no. of dimensions that add up to a sufficiently large proportion of the variance (eg- as %).

Eg:

EVR of PC1 = 0.84

EVR of PC2 = 0.10

EVR of PC3 = 0.05

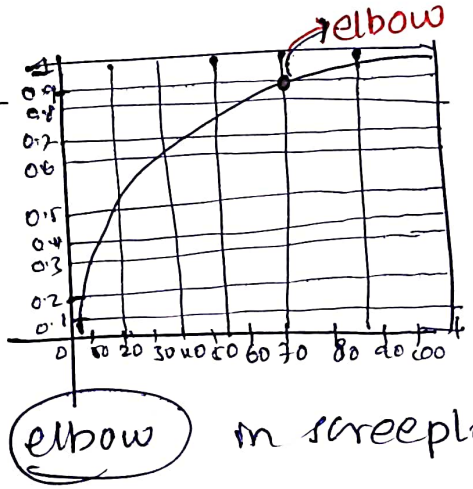
→ Now, if we want to preserve (>90%) variance, then we can just take PC1 & PC2, as they sum up to 0.94, 94%, >90%.

→ Instead if we want to preserve (>95%) of variance, then we should take PC1, PC2 & PC3. as the sum up to 0.99, 99%. >95%.

① Scree plots

→ Scree plots are the graphs that convey how much variance is explained by corresponding principal components.

→ So, by eyeballing the screeplot and looking for a point at which the proportion of variance explained by each subsequent PC drops off. This is often referred as elbow in screeplots.

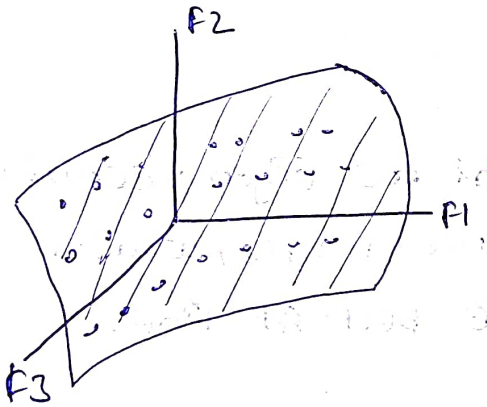


Issues / concerns with PCA

1. Time complexity
   - In particularly, covariance matrix.
   - Usually, it takes $O(d^3)$
   - If no. of features were large, (like 32000 for image) then d^3 is going to be very very large.

   Finding the eigen vectors and eigen values of covariance matrix is the time consuming step.

   Eg: Face recognition (Eigen faces)

2. 

   Data may not necessarily live in a low-dimensional Linear subspa

   → features will not be linearly related.

Notes

- we will have same solution for both the above issues, solving one will solve the other.

Kernel PCA

- Input — $\{x_1, \dots x_n\}$ $x_i \in \mathbb{R}^d$,

Kernel $K: \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$

Step ① compute $K \in \mathbb{R}^{n \times n}$,
where $K_{ij} = K(x_i, x_j)$ $\forall i, j$.

And then we need to center the kernel

Step ② compute $\beta_1, \dots \beta_l$ eigen vectors & $n\lambda_1 \geq \dots n\lambda_l$ eigen values.

~~step~~ and normalize to get, $\alpha_k = \frac{\beta_k}{\sqrt{n\lambda_k}}$ $\forall k$.

Step ③ $w_k = \phi(x) \cdot \alpha_k$

Eigen computation of kernel matrix. Hence, thus defeats the purpose, as it needs $\phi(x)$, which we tried to avoid all along!!

we cannot reconstruct the eigen vectors of the covariance matrix

→ But we can still compute the compressed representation.

$$\phi(x_i)^T \cdot w_k = \phi(x_i)^T \cdot \left(\sum_{j=1}^{n} \phi(x_j) \alpha_{kj} \right) = \sum_{j=1}^{n} \alpha_{kj} \phi(x_i)^T \phi(x_j)$$

$$\phi(x_i)^T w_k = \sum_{j=1}^{n} \alpha_{kj} \cdot K_{ij}$$

→ for downstream task, this is usually good enough.

→ Typically, the reason why we do kernel PCA is to get these projections and throw away original points and only retain these projections along each of these data points.

⭐ This is where, dimensionality Reduction happens.

(eg

→ original datapoint was 100 dimension, but then you only care about top 5.

→ So, we map the 100 dimensional data into a 5 dimensional projection onto these 5 most important directions.

→ That becomes a 5 dimensional representation of your 100 dimensional data

→ once, you learn these representation in a low dimensional space, as just the projection onto these eigen directions you can use these projection values as your data point and work this for a downstream task, (supervised learning task).

modified

Step (3) compute $\sum_{j=1}^{n} \alpha_{kj} \cdot k_{ij} \quad \forall k$

$$x \in \mathbb{R}^d \longrightarrow \left[\sum_{j=1}^{n} \alpha_{1j} \cdot k_{ij}, \sum_{j=1}^{n} \alpha_{2j} \cdot k_{ij} \cdots \sum_{j=1}^{n} \alpha_{lj} \cdot k_{ij} \right]$$

→ It might increase the dimension, nevertheless it will allow you to capture more complicated relationships.

Note:

→ After step 2, we have to center the kernel.

→ Given, $K \in \mathbb{R}^{n \times n}$

$K_{ij} = K(x_i, x_j) \quad \forall i, j$

create a new kernel K^c — centered.

$$K^c_{ij} = K_{ij} - \theta_i 1_j^T - 1_i \theta_j^T + P 1_i \cdot 1_j^T$$

where,

$$\theta_i = \frac{1}{n} \sum_{k=1}^{n} K_{ik} \quad \forall i$$

$$P = \frac{1}{n^2} \sum_{i,j} K_{ij}$$

→ Now, we can say we are able to solve both the issues by using kernel PCA.