Advanced Machine Learning and Deep Learning

Dr. Mohammad Pourhomayoun
Assistant Professor
Computer Science Department
California State University, Los Angeles

Large-Scale Data

Large-Scale Machine Learning

- There are several approaches to handle **learning from Big Data**:
  1. Modifying the learning algorithms.
     - E.g. Stochastic Gradient Descent (covered in CS4662)
     - E.g. Mini-Batch Gradient Descent (covered in CS4662)
  2. Parallel Computing and MapReduce (covered in CS4661).
  3. Dimensionality Reduction (CS4662).
  4. Sampling the big dataset and only using the samples (Simplest, but not the best approach, covered in CS4661).

Dimensionality Reduction

Dimensionality Reduction

- **Dimensionality Reduction** is an important research topic in data science and machine learning.
- Dimensionality Reduction is the process of representing the data using a smaller number of features (i.e. smaller dimensionality) in an efficient way.
- The **main goal of Dimensionality Reduction is to keep the most useful part of the data, and/or eliminate redundant and/or useless part of the data**.
- **REMEMBER**: No matter how much you increase the size of memory or increase the computational power, we still cannot keep up with the data that we would like to collect and process. So, dimensionality reduction can be very helpful!

Applications of Dimensionality Reduction

- Data Compression
  - Reduce Memory to store data
  - Reduce Bandwidth to transmit data
- Visualization
  - Move the data to Low-Dimensional Space (2D or 3D) so that we can Visualize it!
- Dimensionality Reduction for Machine Learning (the main application for our class)
  - Reduce Computation Complexity of algorithms and speed up the Learning Process
  - Remove Redundant part of data
  - Remove Useless or harmful part of data

Data Compression

- Two Types of Data Compression:

1. **Lossless Compression:** Only Remove the Redundant part of data

- Zip, Rar, …

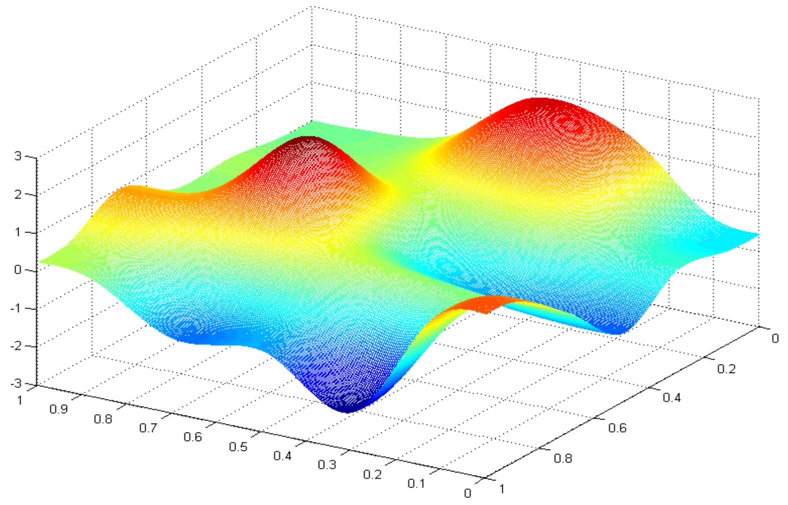
2. **Lossy Compression**: Remove the Useless or Less important part of data

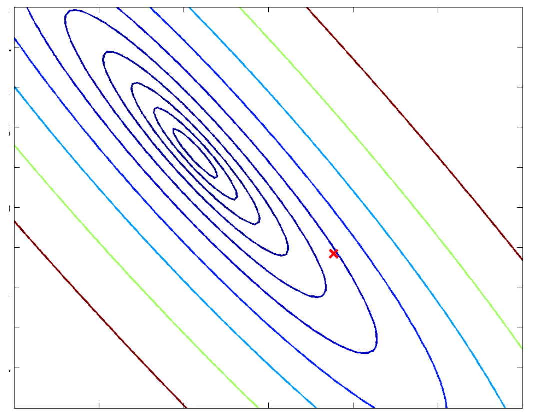
- Images: jpg, png, …
- Audio: mp3, …
- Video: mp4, avi, mkv, …

Dimensionality Reduction for Visualization

- **Move the data to Low-Dimensional Space (2D or 3D) so that we can Visualize it!**

?





4D **3D** **2D**

Dimensionality Reduction For Machine Learning

Dimensionality Reduction

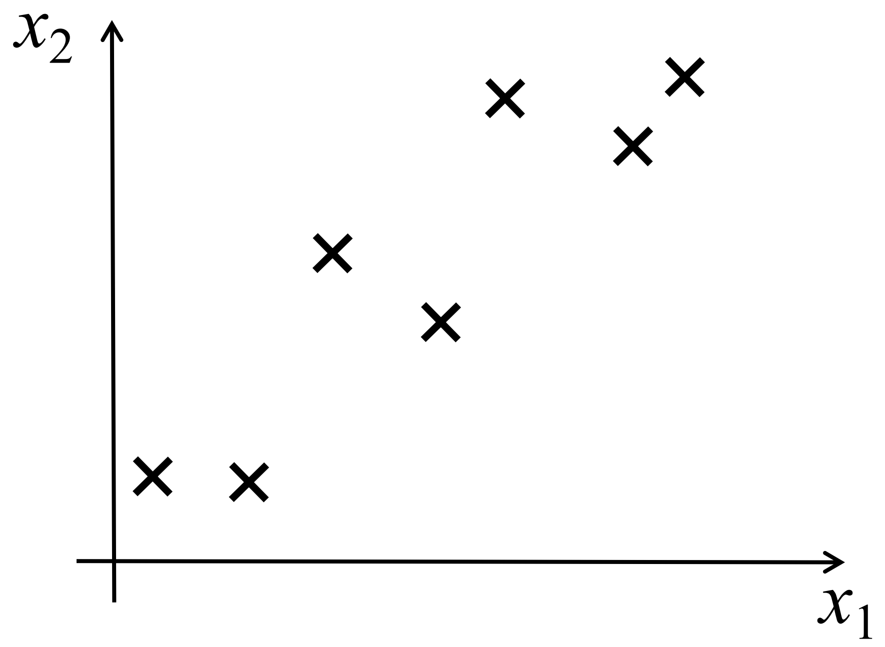
- Dimensionality Reduction is the process of representing the data using a smaller number of features (i.e. smaller dimensionality) in an efficient way.
- The **main goal of Dimensionality Reduction is to keep the most useful part of the data, and/or eliminate redundant and/or useless part of the data**.
- They are two main approaches for dimensionality reduction:
  - **Feature Selection**
  - **Feature Extraction**

Dimensionality Reduction

- Feature Selection algorithms try to find **the best subset of the original features that provides the most useful information** for prediction.
  - **Feature Selection does not change the original features. It just removes redundant features and/or useless features from the feature set.**

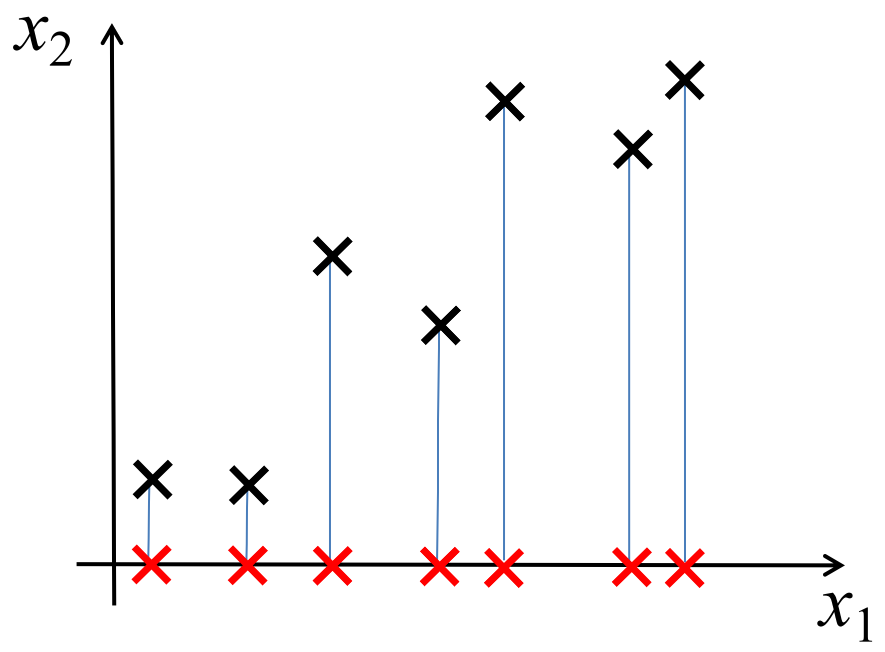
- Feature Extraction algorithms try to **transform** the data from its original high-dimensional space to a new space with fewer dimensions in which the information will concentrate in a number of highly Uncorrelated features.
  - **Feature Extraction changes the original data. The results (new features) are completely different values in another space, which represents the original data in a new and efficient way.**

Example: Reduce Dimensionality from 2D to 1D

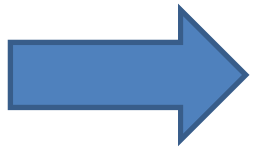


- How to transform it to 1D?
- We need to get rid of one of the axes, but how?
- Any idea?
- How about **Projection**?

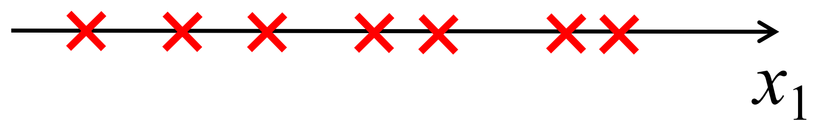
Example: Reduce Dimensionality from 2D to 1D



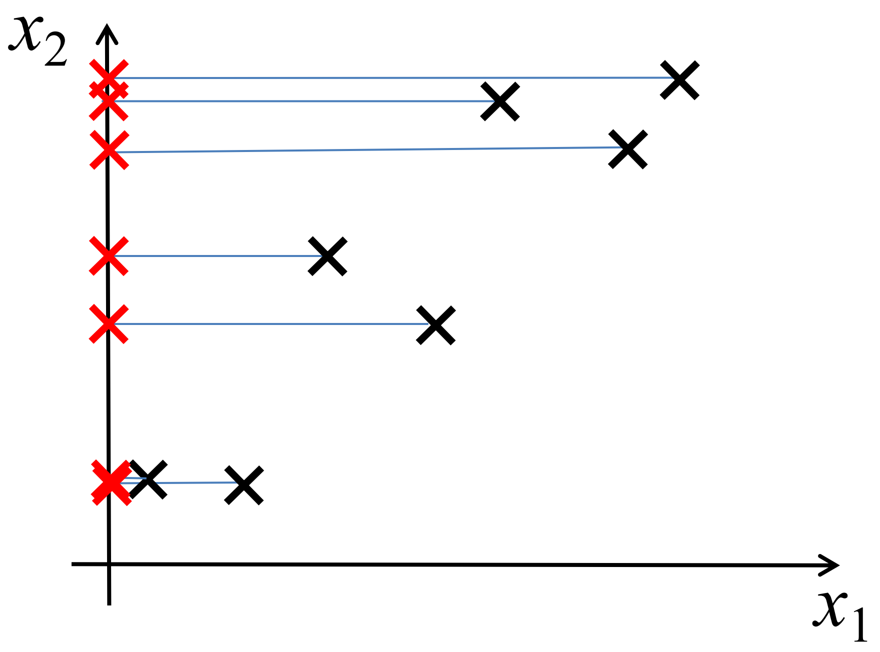
- How about **Projection** on x_1?
- Is this a good approach?
- This is technically a **feature selection**, we did not extract a new feature. This way, we just threw away x_2 , and only kept x_1!
- Big Loss: we completely lost the x_2 feature!



New Data:

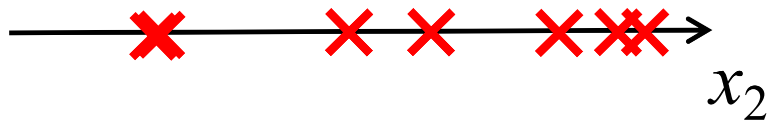


Example: Reduce Dimensionality from 2D to 1D

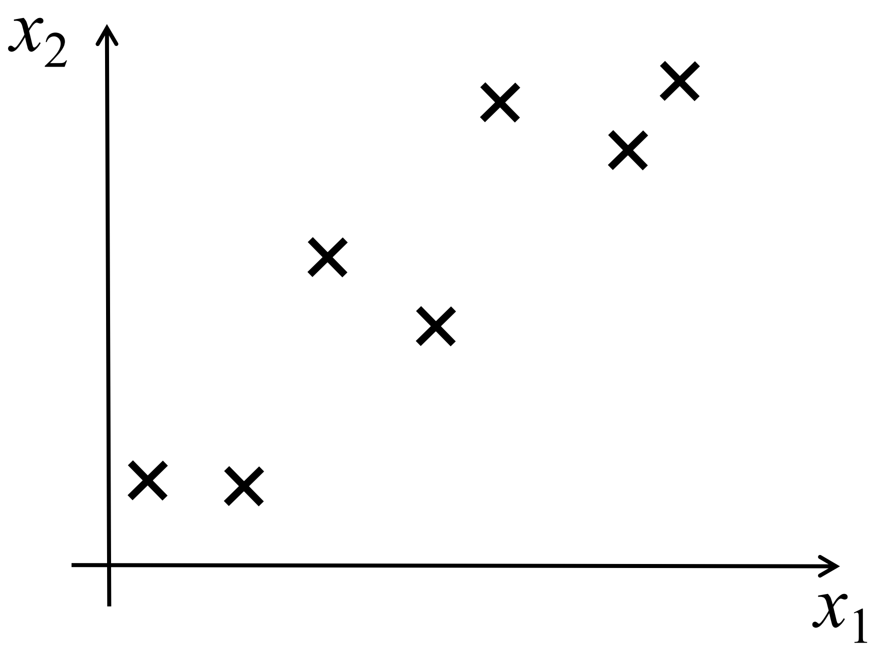


- How about **Projection** on x_2?
- This is again a **feature selection**: we did not extract a new feature. We just threw away x_1 , and only kept x_2!
- Big Loss: we completely lost the x_1 feature!

New Data:

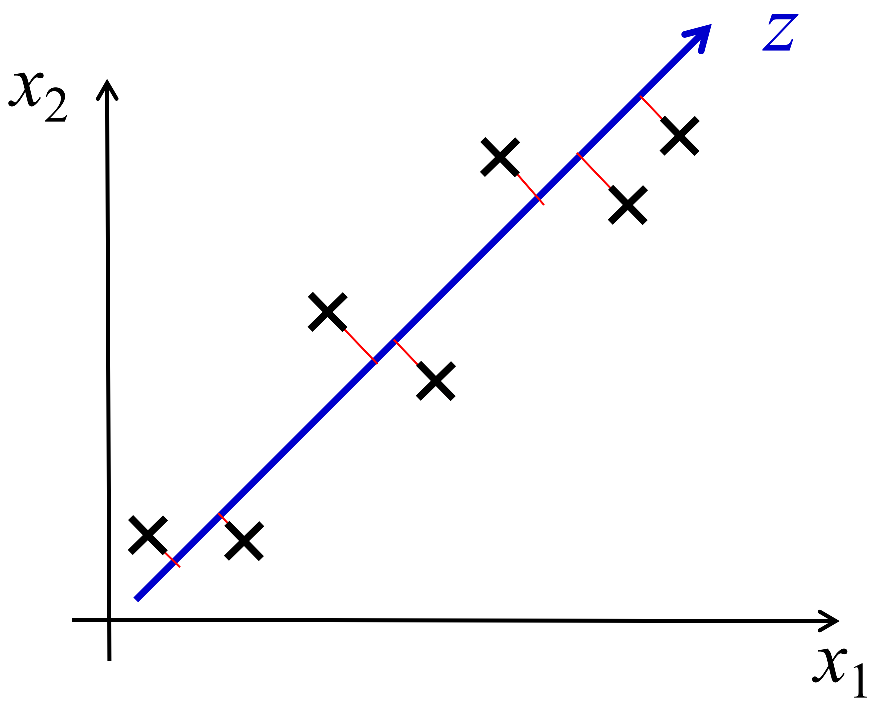


Example: Reduce Dimensionality from 2D to 1D



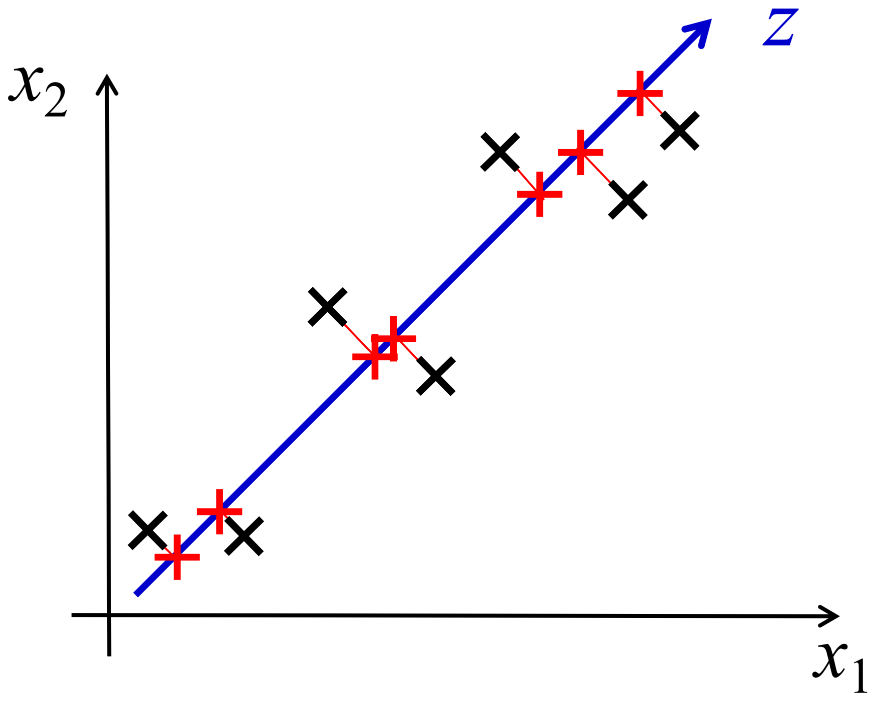
- Any smarter way to do this?
- Any smarter **Projection** so that we can keep the information of both x_1 and x_2?

Example: Reduce Dimensionality from 2D to 1D



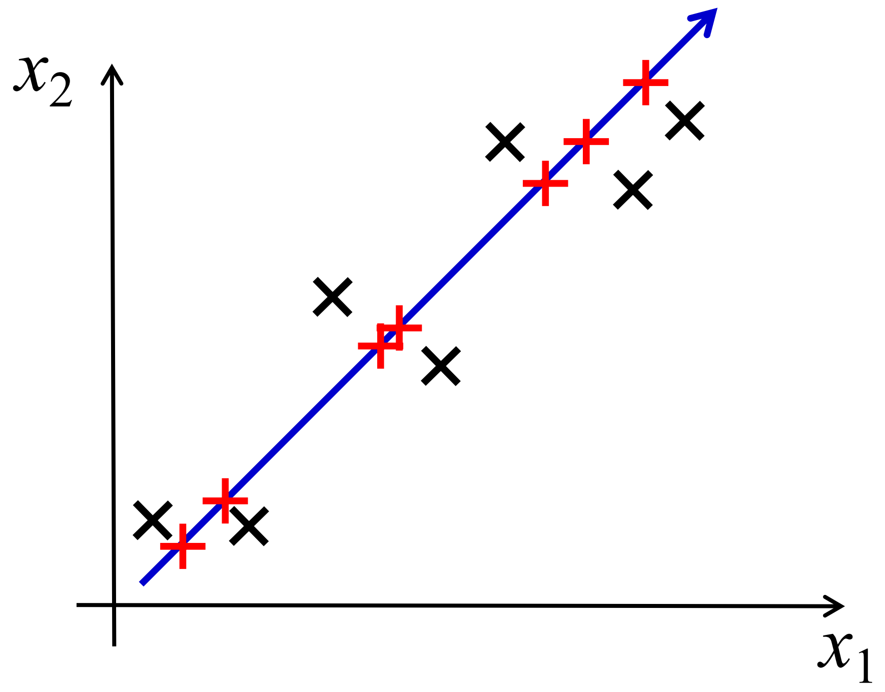
- Any smarter way to do this?
- Any smarter **Projection** so that we can keep the information of both x_1 and x_2?
- How about the **Projection** on this new blue axis?

Example: Reduce Dimensionality from 2D to 1D



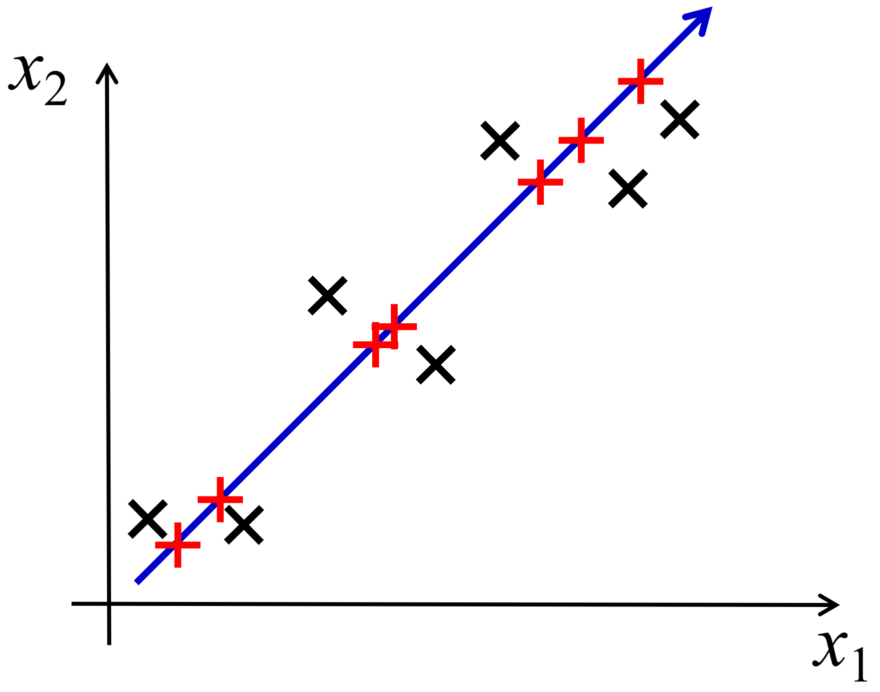
- Any smarter way to do this?
- Any smarter **Projection** so that we can keep the information of both x_1 and x_2?
- How about the **Projection** on this new blue axis?

Example: Reduce Dimensionality from 2D to 1D



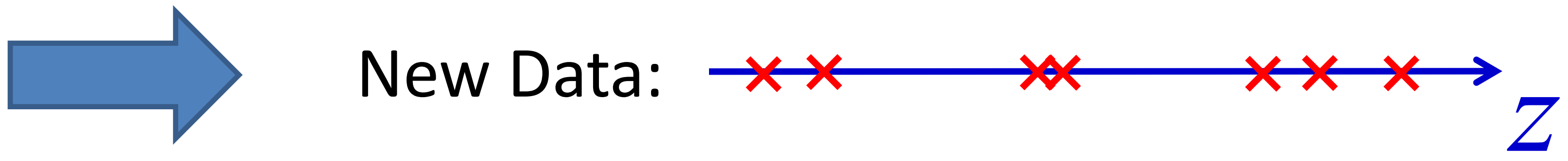
- Any smarter way to do this?
- Any smarter **Projection** so that we can keep the information of both x_1 and x_2?
- How about the **Projection** on this new blue axis?

Example: Reduce Dimensionality from 2D to 1D



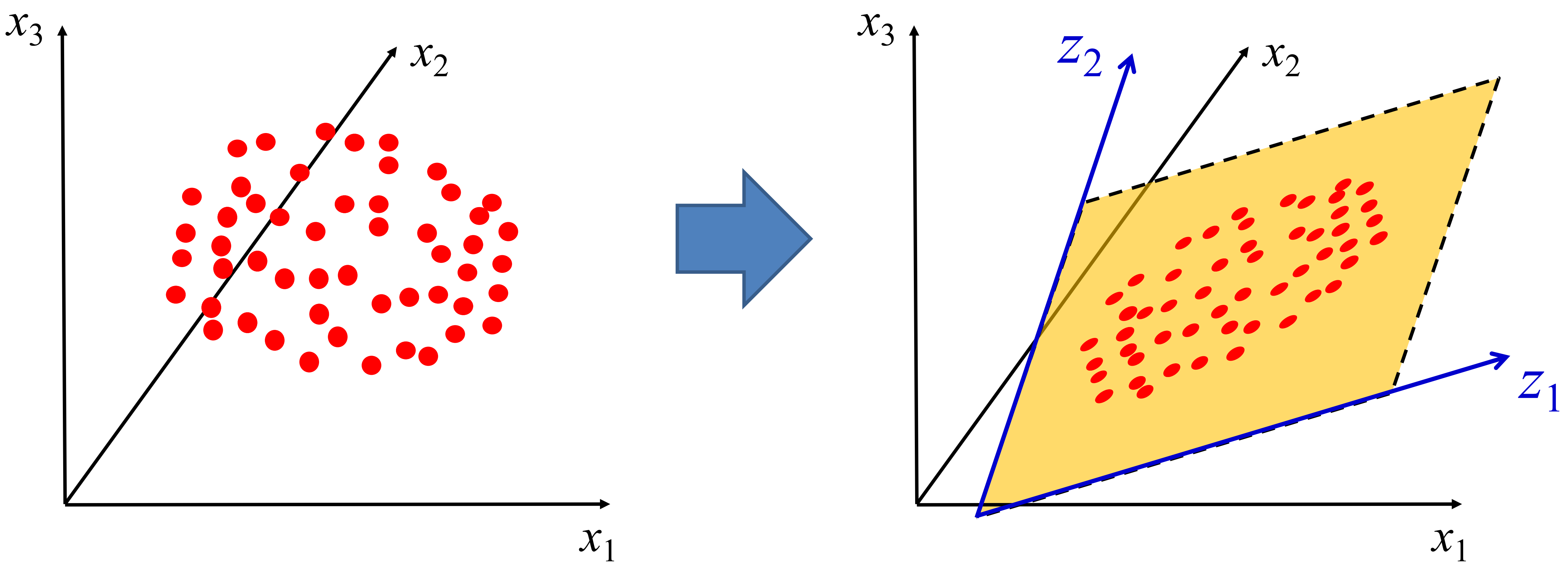
- Z **is a new "compact" feature that includes most of the information of both x_1 and x_2!!**

- As we see, the ***projection error*** (the distance between the original points and the projections) is very small in this method!

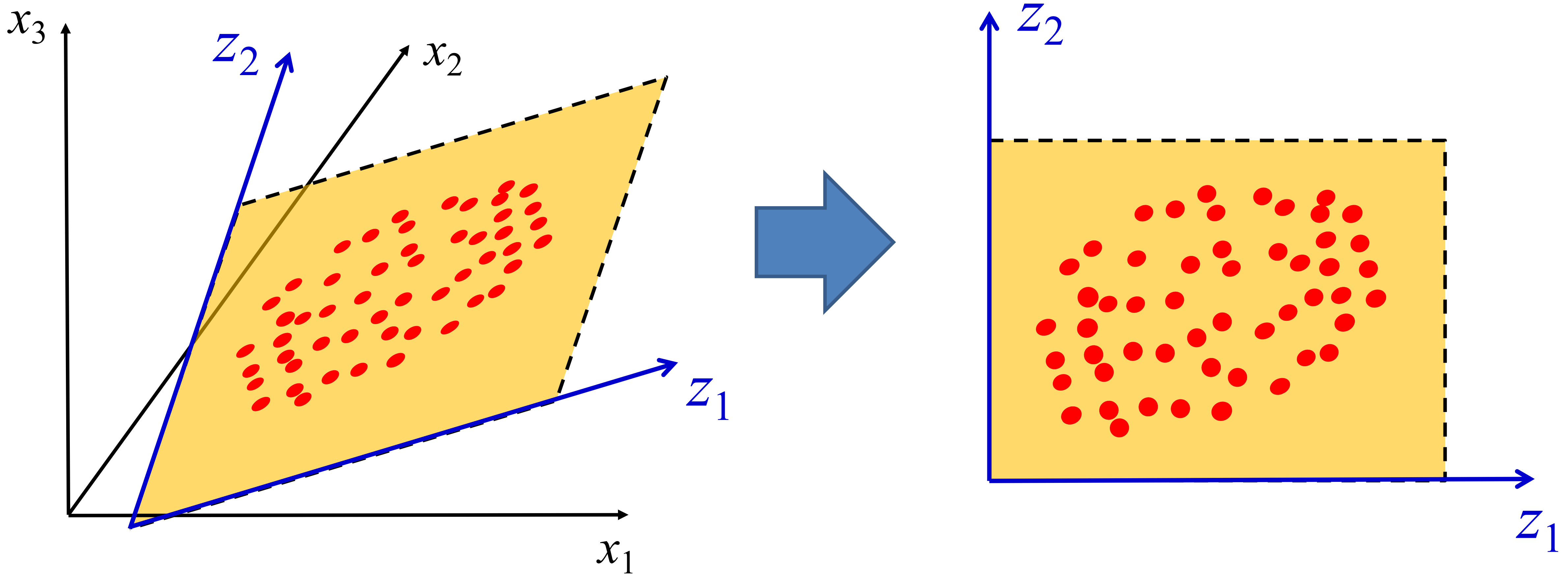


New Data:

Example: Reduce Dimensionality from 3D to 2D



Example: Reduce Dimensionality from 3D to 2D



Big Question!

- **Big Question:** How to find the best new axis (or axes), onto which to project the data, and have the minimum amount of information loss?
- In other word, what is the best axis (or axes) for projection that minimizes the projection error?

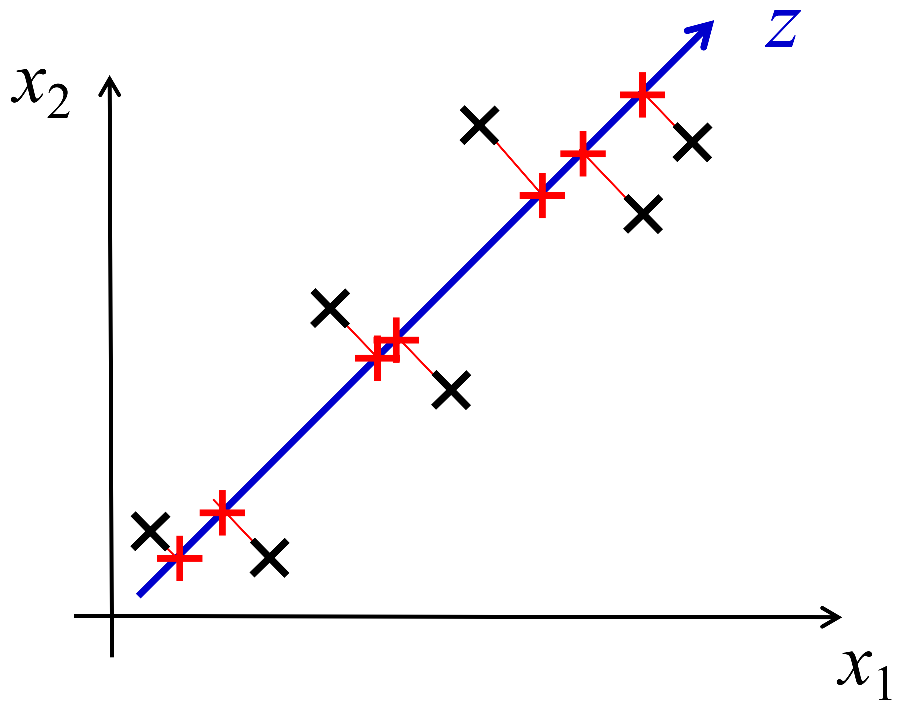
- **Answer: Principal Component Analysis (PCA)** does this for us!

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

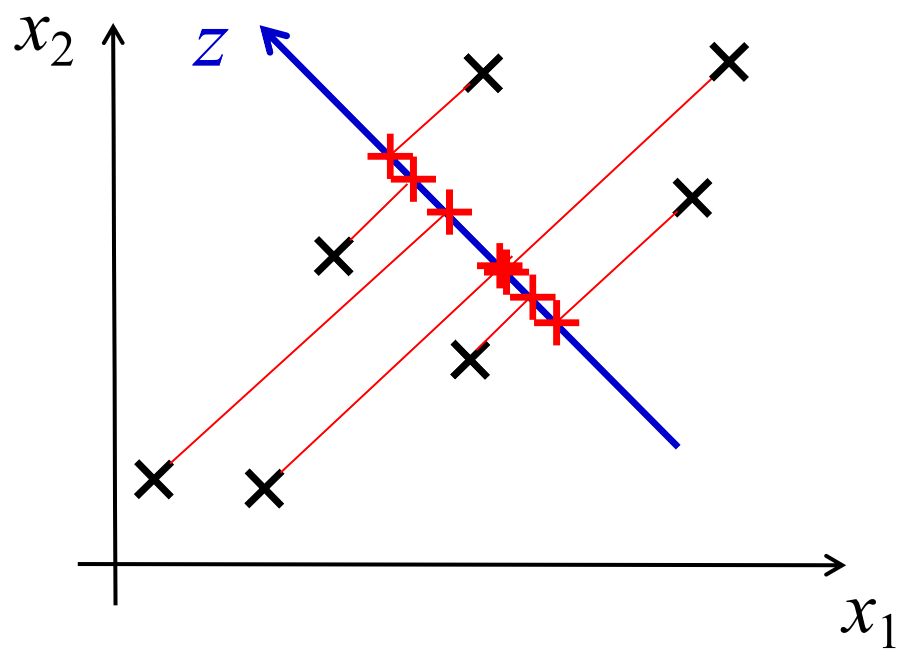
- We would like to find the best axis (or axes) z, that minimizes the projection error, (error is the distance between **original points (black)** and their **projections (red)**)

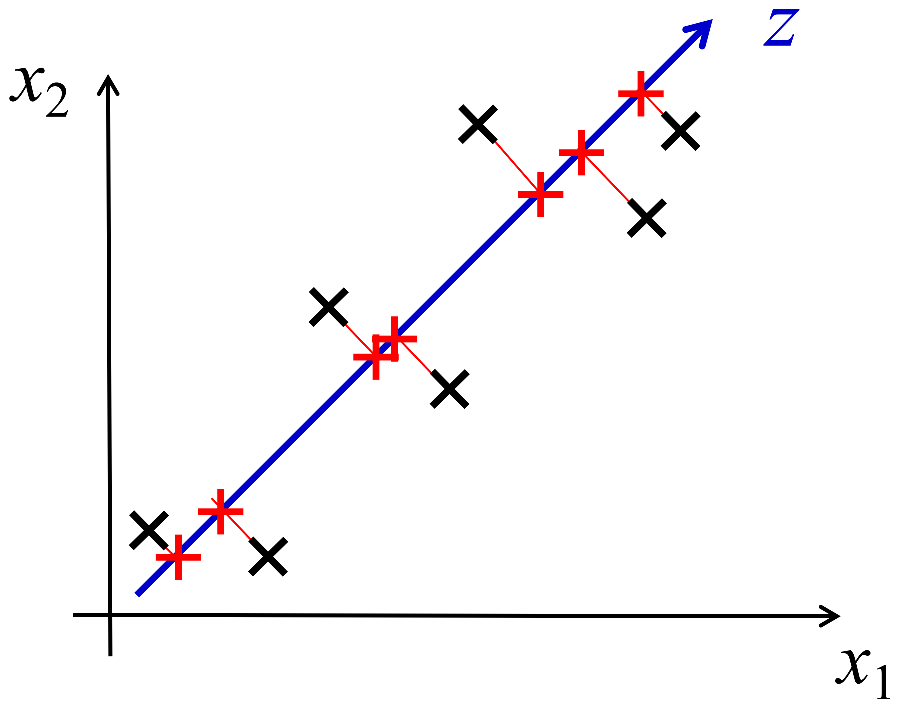
- **PCA solves this problem!**



Principal Component Analysis (PCA)

- **Which Projection is better?**
- **Answer: The axis z in the right figure is better because the projection error is less!**

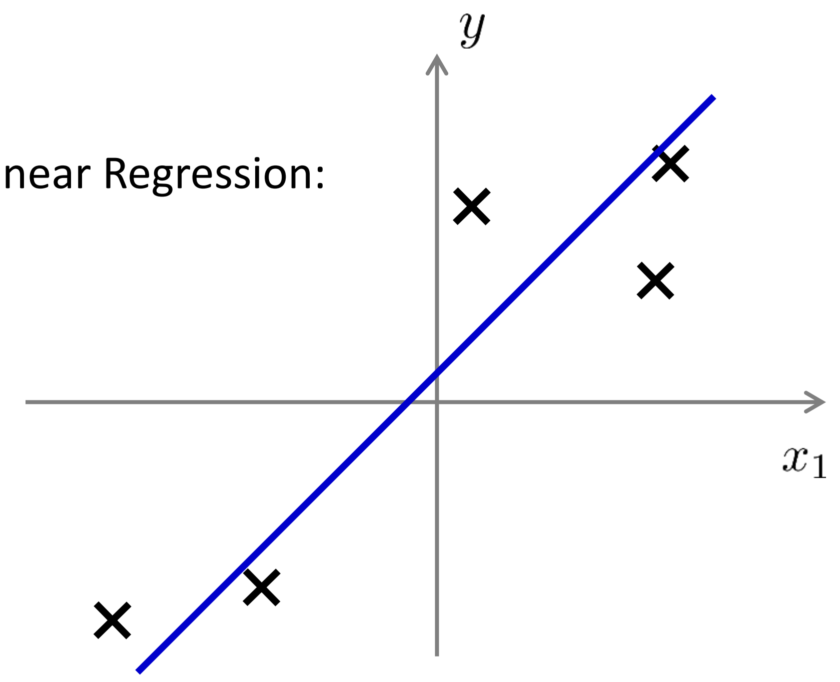




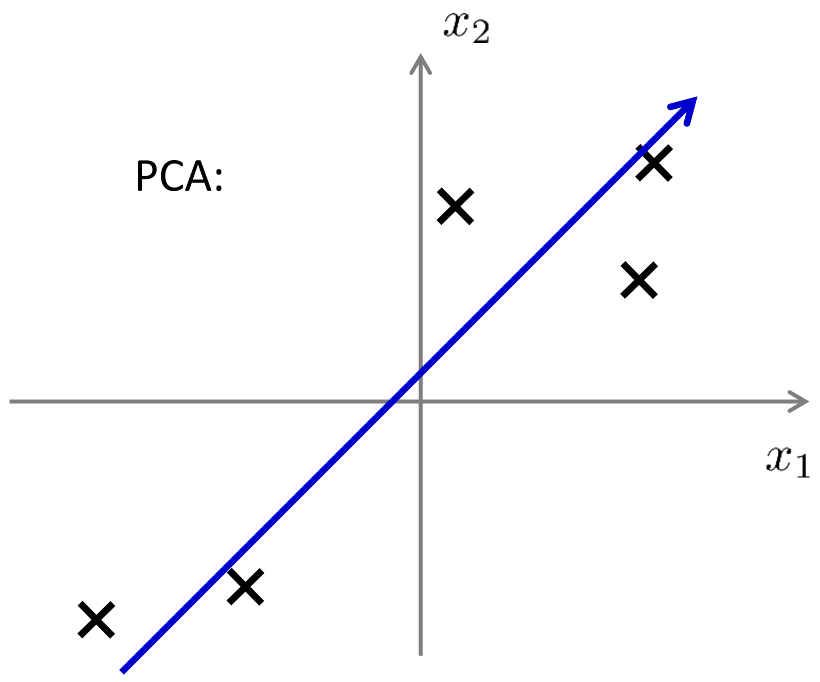
Difference between PCA and Linear Regression?

- **Question**: So, **PCA** and **Linear Regression** both tries to find the **best fit line** to minimize the error between original points and the line. What is the difference between PCA and Linear Regression?

Linear Regression:

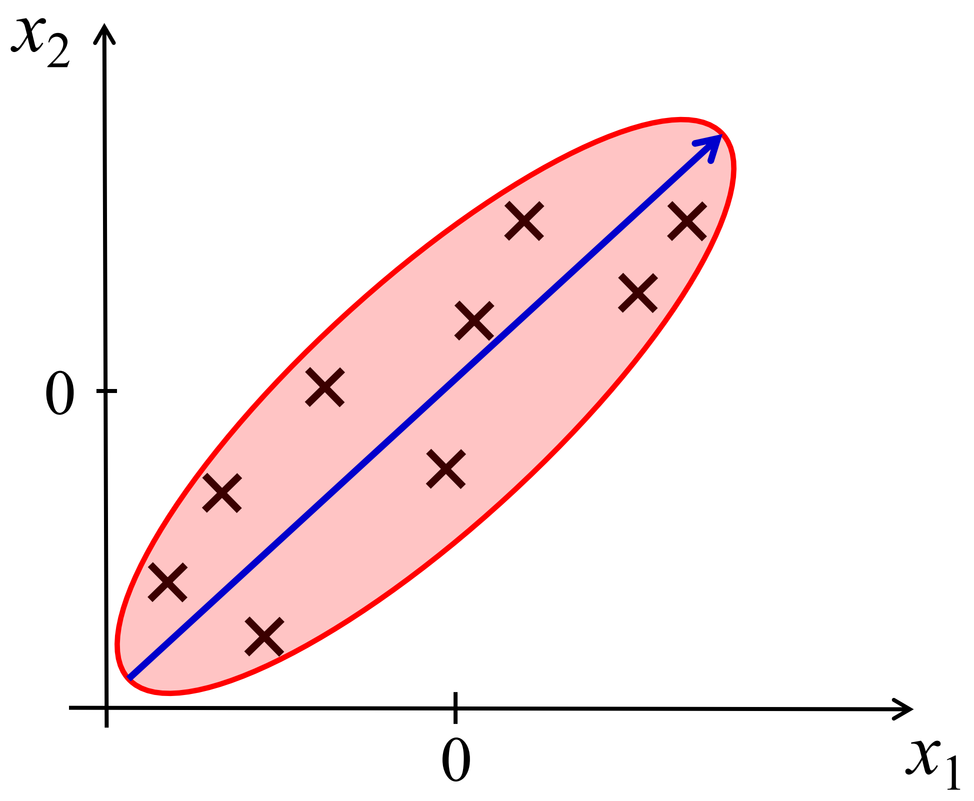


PCA:



Principal Component Analysis (PCA)

- We would like to find the best axis (or axes) z, that minimizes the projection error (the distance between **original points (black)** and their **projections on the line**).
- **To do this, we should study the data distribution and variation patterns.**
- PCA solve this problem by **finding the directions of maximal variance**!

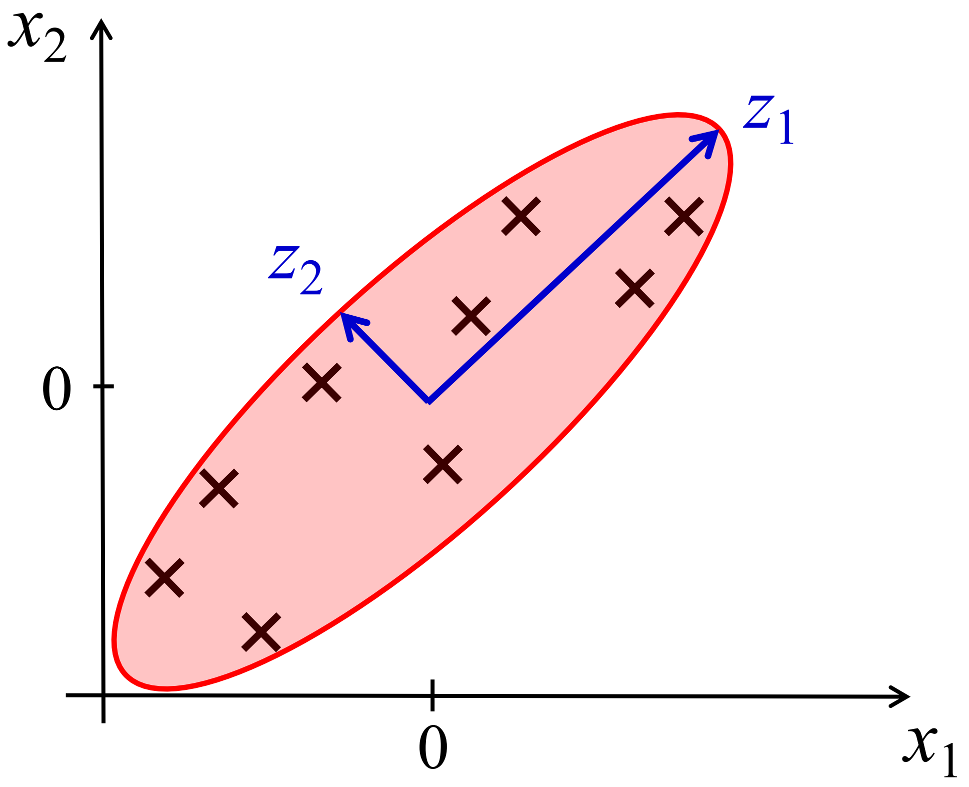


Definition

- **Principal component analysis (PCA)** uses an ***orthogonal transformation*** to convert the data ***of correlated features*** into a set of new data samples ***of linearly uncorrelated features.***
- These new uncorrelated features are called ***principal components***.

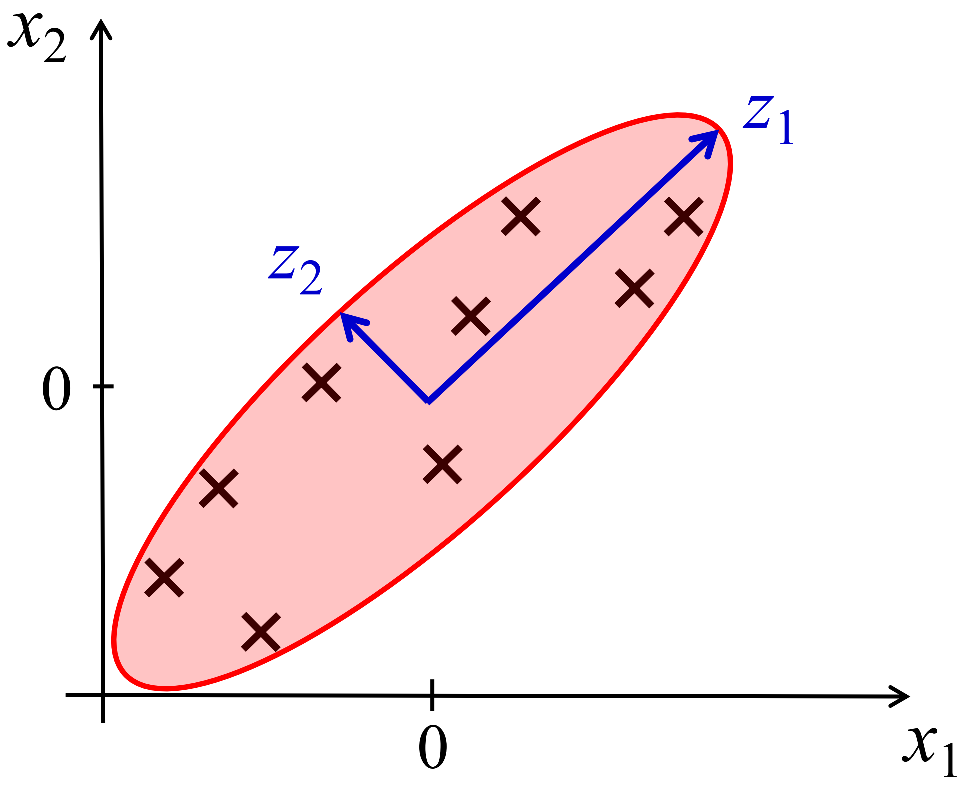
Principal Component Analysis (PCA)

- PCA can be interpreted as ***fitting an n-dimensional ellipsoid to the data***.
- In this case, ***each axis of this ellipsoid represents a principal component*** (axes of our new feature space).
- If an axis of the ellipsoid is small (e.g. z_2), then the variance along that axis is also small, and by omitting that axis, we lose only a small amount of information.

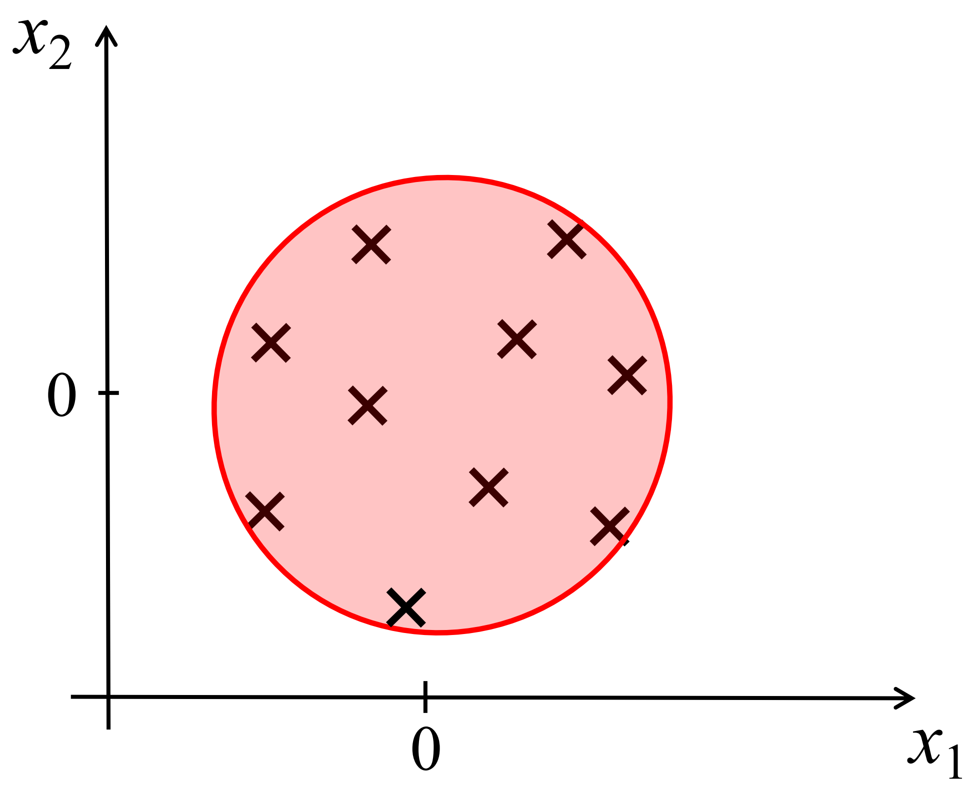


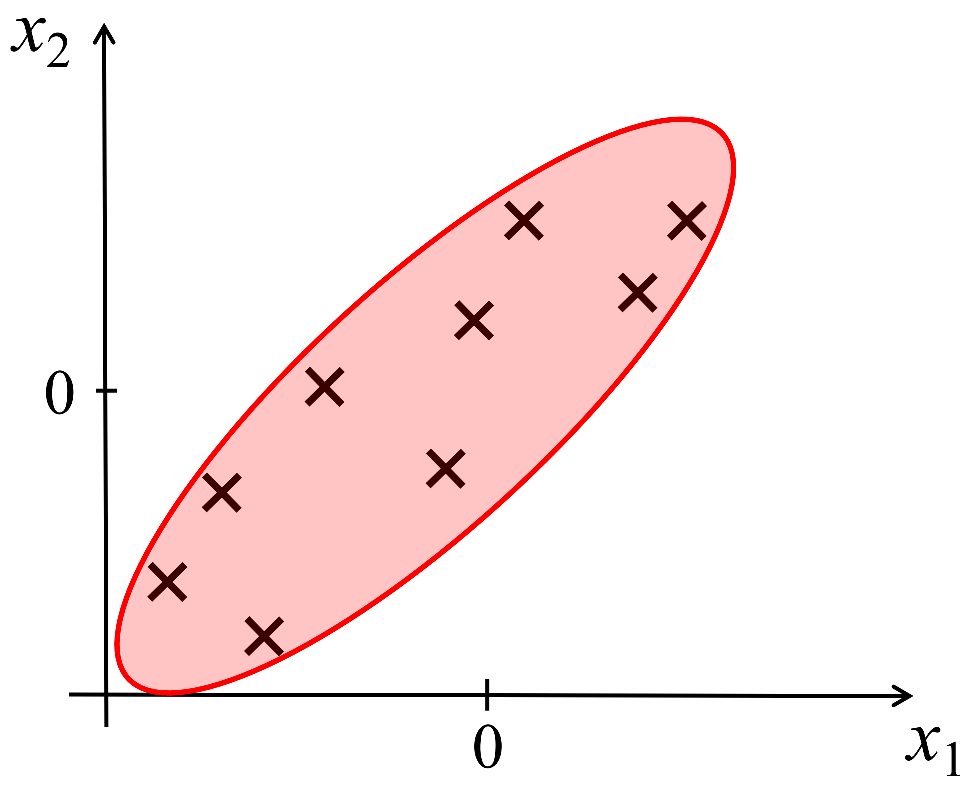
Principal Component Analysis (PCA)

- PCA can be interpreted as ***fitting an n-dimensional ellipsoid to the data***.
- In this case, ***each axis of this ellipsoid represents a principal component*** (axes of our new feature space).
- If an axis of the ellipsoid is small (e.g. z_2), then the variance along that axis is also small, and by omitting that axis, we lose only a small amount of information.
- In summary, we have to find axes of the ellipsoid (z_1 and z_2), and then ***project*** the data samples on z_1 (i.e. we can omit z_2).



Question: Which Dataset is More Compressible?





Review: Matrix to Vector Multiplication

- **<u>Matrix as Transform:</u>** Our main view of matrices will be as "***operators***" that transform one vector into another vector.

$$\underset{m\times 1}{\boldsymbol{y}} = \underset{m\times n}{\boldsymbol{A}}\,\underset{n\times 1}{\boldsymbol{x}}$$

- **Special Case:** If the mapping matrix ***A*** is ***square,*** then the space that vectors get mapped to has the same dimension.

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow \begin{aligned} y_1 &= ax_1 + bx_2 \\ y_2 &= cx_1 + dx_2 \end{aligned}$$

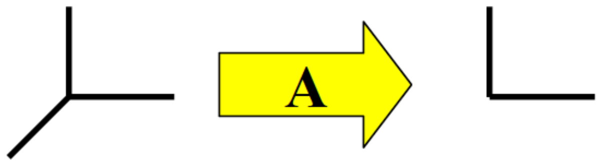
Review: Matrix to Vector Multiplication

- Consider a 2x3 matrix below. We can use that matrix to ***transform*** the ***3-dimensional*** vector $\boldsymbol{x}$ into a ***2-dimensional*** vector $\boldsymbol{y}$:

$$\underbrace{\boldsymbol{y}}_{m\times 1} = \underbrace{\boldsymbol{A}}_{m\times n}\underbrace{\boldsymbol{x}}_{n\times 1}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & -1 \\ 3 & -2 & 5 \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} 1 \\ 3 \\ 9 \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & 2 & -1 \\ 3 & -2 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 9 \end{bmatrix} = \begin{bmatrix} -2 \\ 33 \end{bmatrix}$$



Review: Eigenvalues and Eigenvectors

- **Question:** For a given nxn matrix A, which vectors get mapped into being almost themselves???
- More precisely, which vectors get mapped to a scalar multiple of themselves???
- Even more precisely, which vectors v satisfy the following:

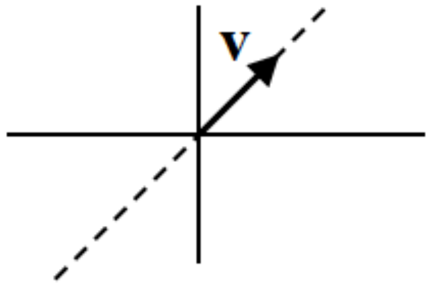
$$\boxed{Av = \lambda v}$$

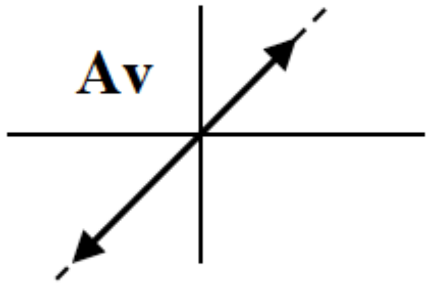
- These vectors v are “special” and are called the ***eigen-vectors*** of A.
- The scalars λ is the corresponding ***eigen-values***.

Review: Eigenvalues and Eigenvectors

$$\boxed{Av = \lambda v}$$

- These vectors v are called the ***eigenvectors*** of ***A***.
- The scalars λ is the corresponding ***eigenvalues***.
- **In other word, Transformation using matrix *A* on its eigenvectors acts just like Multiplying these vectors by a Scalar!**





Principal Component Analysis (PCA)

- **How can PCA find the best axes?**
- **Answer**: ***It turns out that the best axes (axes of the ellipsoid, i.e. principal components) are Eigen-Vectors of the covariance matrix (C) of the data!*** (refer to lecture13 for eigen-vector)
- Since the ***covariance matrix is symmetric***, the eigenvectors are ***orthonormal*** (refer to lecture13 for eigenvector properties!), and we can decompose C as:

$$C = U\,\Lambda\,U^T$$

- where U is a matrix with C's eigenvectors as columns, and Λ is a diagonal matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ on main diagonal and 0 everywhere else.

Principal Component Analysis (PCA)

- **How can PCA find the best axes?**
- **Answer**: ***It turns out that the best axes (axes of the ellipsoid, i.e. principal components) are Eigen-Vectors of the the covariance matrix (C) of the data!***

- **The eigenvectors are orthonormal directions of max variance. Associated eigenvalues equal variance in these directions!**
- **The proportion of the variance that each eigenvector (each axis) represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues.**

PCA Formulation

- Suppose that we have n data samples, and d features:
  - $\boldsymbol{X}$ is an (n x d) matrix (Original Feature Matrix).
  - $\boldsymbol{Z}$ is an (n x k) matrix (Feature Matrix after Dimensionality Reduction).
  - $\boldsymbol{P}$ is an (d x k) matrix (**k << d**) (columns are k Principal Components).
  - PCA Formulation: $\boldsymbol{Z}_{(n \times k)} = \boldsymbol{X}_{(n \times d)} \boldsymbol{P}_{(d \times k)}$

$$\begin{bmatrix} \boldsymbol{Z} \end{bmatrix}_{n\times k} = \begin{bmatrix} \boldsymbol{X} \end{bmatrix}_{n\times d} \begin{bmatrix} \boldsymbol{P} \end{bmatrix}_{d\times k}$$

PCA Formulation (optional)

- Given n training points with d features:
- $x_j^{(i)}$: jth feature of the ith data sample
- μ_j :mean of the jth feature

Variance of 1st feature:

$$\sigma_1^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_1^{(i)} - \mu_1\right)^2$$

Variance of 1st feature:
(assuming zero mean)

$$\sigma_1^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_1^{(i)}\right)^2$$

PCA Formulation (optional)

- Given n training points with d features:
- $x_j^{(i)}$: jth feature of the ith data sample
- μ_j :mean of the jth feature

Covariance of 1st and 2nd feature:
(assuming zero mean)

$$\sigma_{12} = \frac{1}{n} \sum_{i=1}^{n} x_1^{(i)} x_2^{(i)}$$

- Zero Covariance → uncorrelated features
- Large Covariance → correlated features → redundant features

PCA Formulation (optional)

- Covariance matrix is a d x d matrix including the features' variances/covariances:
  - i^{th} diagonal entry equals variance of i^{th} feature
  - ij^{th} entry is covariance between i^{th} and j^{th} features

d x d Covariance matrix:
(assuming zero mean)

$$C = \frac{1}{n}\boldsymbol{x}^T\boldsymbol{x}$$

- Zero Covariance → uncorrelated features
- Large Covariance → correlated features → redundant features
- We want $\boldsymbol{Z}$ to have uncorrelated features. Thus, all off-diagonals in $\boldsymbol{C}_Z$ should be zero!

PCA Formulation (optional)

- ***Eigen Decomposition*** of the Covariance Matrix C:

$$\boldsymbol{C} = \boldsymbol{U} \Lambda \boldsymbol{U}^{\boldsymbol{T}} \qquad \boldsymbol{U} = \begin{bmatrix} \boldsymbol{Z}_1 & \boldsymbol{Z}_2 & \cdots & \boldsymbol{Z}_d \end{bmatrix} \qquad \Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & \ddots & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \lambda_d \end{bmatrix}$$

- where $\boldsymbol{U}$ is a matrix with $\boldsymbol{C}$'s eigenvectors as columns, and $\boldsymbol{\Lambda}$ is a diagonal matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ on main diagonal and 0 everywhere else.

PCA Formulation

- **In summary, Here is the Standard Procedure for PCA Dim. Reduction:**
  1) Having the feature matrix $\boldsymbol{X}$, find the Covariance Matrix $\boldsymbol{C}$
  2) Perform Eigen-Decomposition: $\boldsymbol{C = U \Lambda U^T}$
  3) Now, we have d eigen-vectors as column of matrix $\boldsymbol{U}$, select k of them (associated with k largest eigen-values in $\boldsymbol{\Lambda}$) as main principal components for Dim Reduction.
  4) Put the above k eigen-vectors as columns of a new matrix $\boldsymbol{P}$.
  5) Project the data into low-dimensional space by multiplying your original feature matrix to the matrix $\boldsymbol{P}$: $\boldsymbol{Z = XP}$

Choosing New Dimensionality k

- **Question:** How should we select the dimensionality of the new data (the number of new features k)?
- **Answer:** The new dataset should include '**most**' of the variance in the data. since eigenvalues (in Eigen Decomposition) are variances in the directions specified by eigenvectors, we can choose k such that we retain a big fraction of the variance (***95% to 99% is recommended***).
- **Thus, select k , such that:**

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} > 95\% \text{ or } 99\%$$

PCA in Python

- **from sklearn.decomposition import PCA**
- **k = 50** # (*k* is the number of components (new features) after dimensionality reduction)
- **my_pca = PCA(n_components = k)**

  # X_Train is feature matrix of training set before dimensionality reduction,

  # X_Train_New is feature matrix of training set after dimensionality reduction:
- **X_Train_new = my_pca. fit_transform(X_Train)**
- **X_Test_new = my_pca. transform(X_Test)**

Important Practical Tip about PCA

- **Before applying PCA, we MUST preprocess data so that all features have zero mean!**
  - In Python, you can use ***preprocessing.scale*** on your feature table to perform both zero-meaning and normalization.
- Remember that any dimensionality reduction can throw away some part of your data. Thus, sometimes if the size of your dataset is small, you do not need to do dimensionality reduction.

Thank You!

Questions?