Lecture 11: Dimensionality Reduction

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/VE445/index.html

Outline

- Motivation
- PCA
- SVD
- Autoencoder
- Feature selection

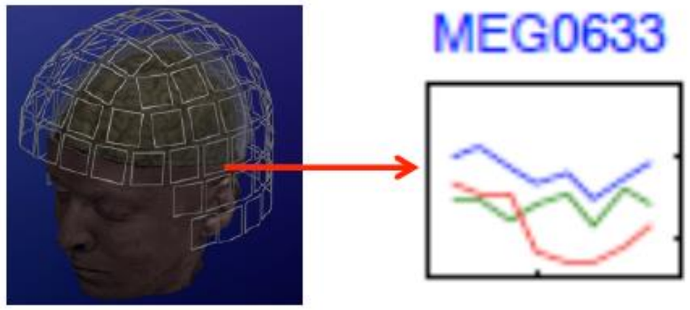
Motivation

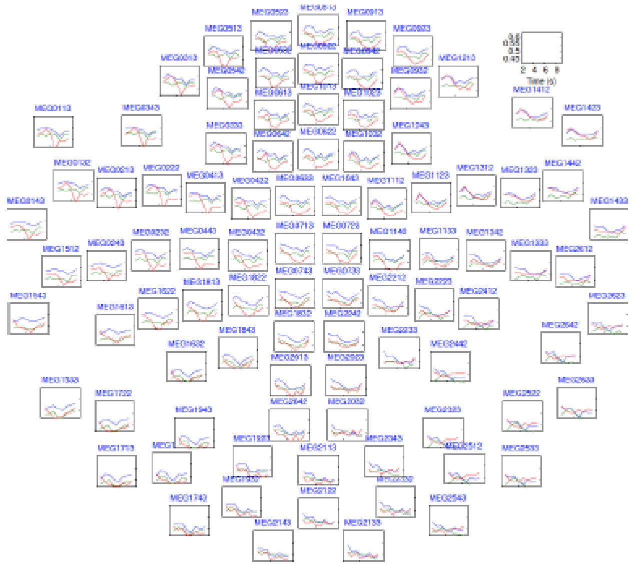
- Suppose we want to predict the health condition of some students, and the features for the students includes:
  - Weight in kilogram
  - Height in inch
  - Height in cm
  - Hours of sports per day
  - Favorite color
  - Scores in math

- Some features are irrelevant, such as favorite color and scores in math
- Some features are redundant, such as height in inch and cm

High dimensional data

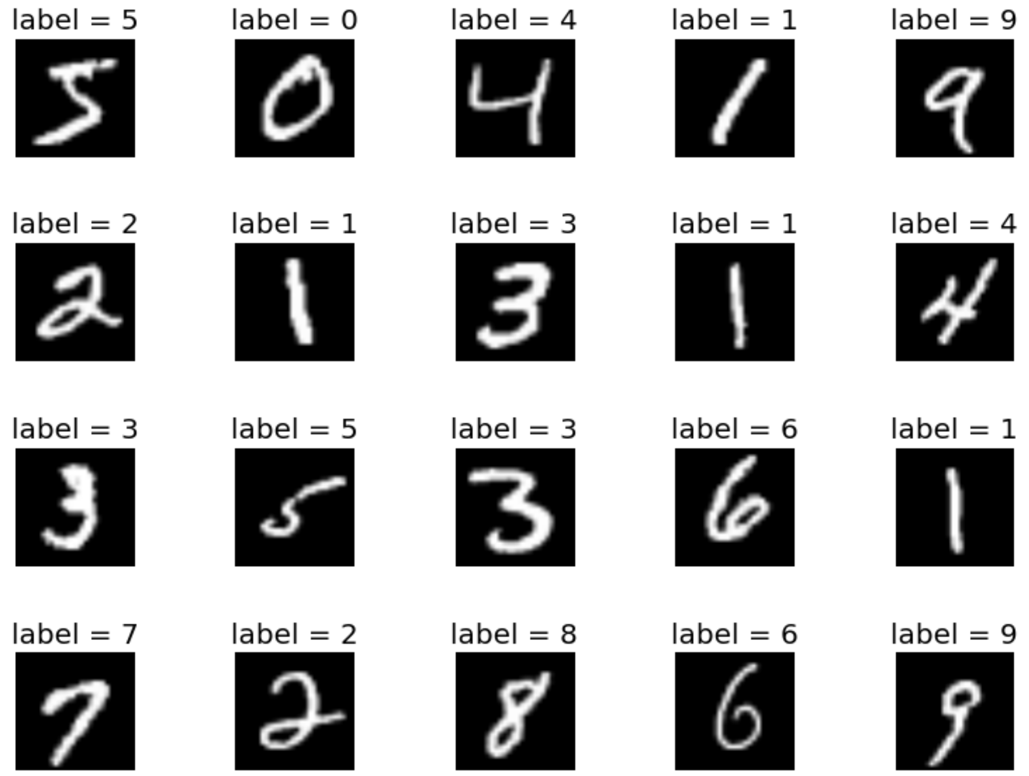
- In the era of big data, the dimensionality increase dramatically
  - E.g. there are many features for the electroencephalogram data

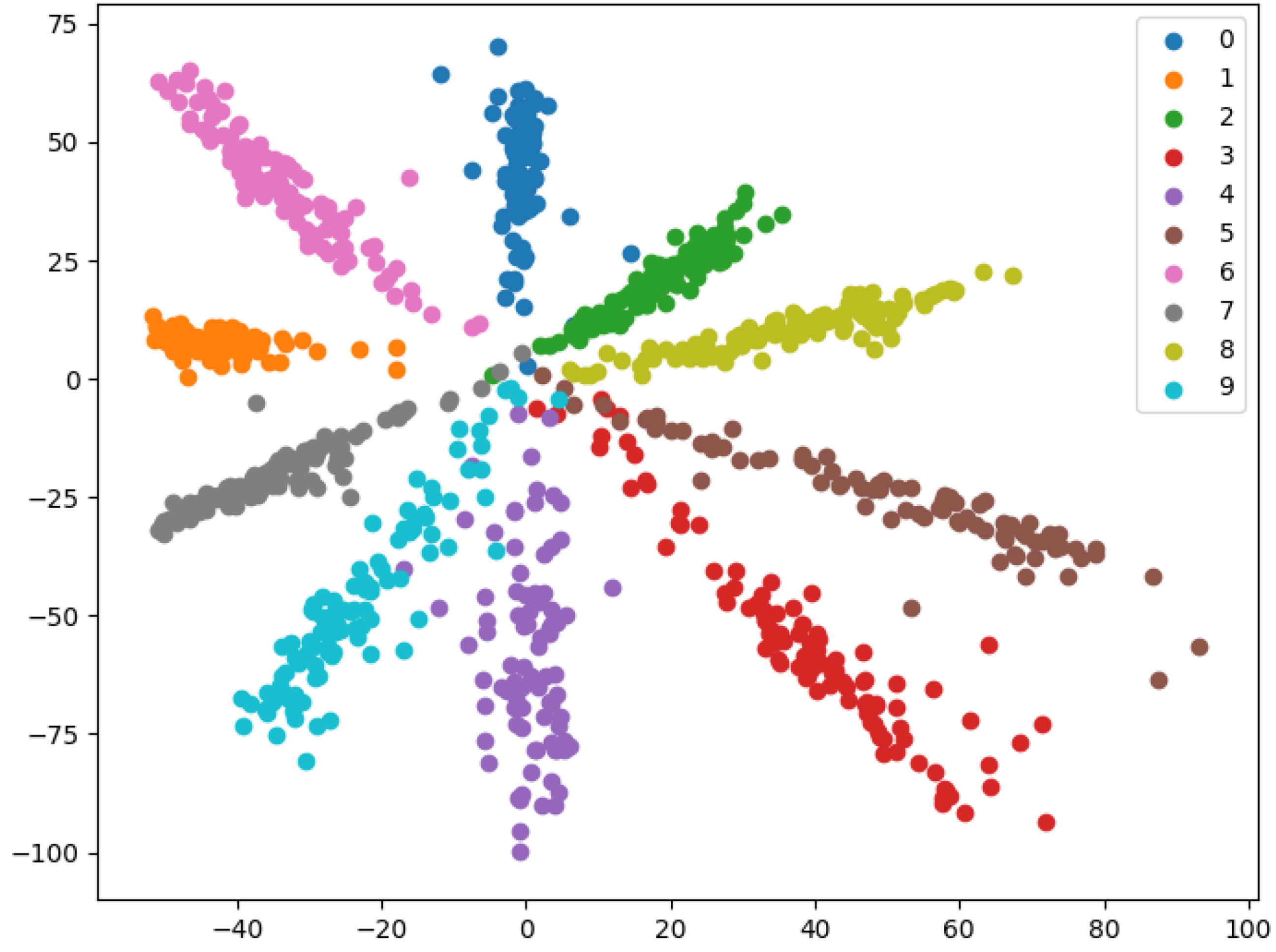




- It become very important to reduce the dimensionality, or select the most important features, or find the most representative features

Example – MNIST dataset





Some methods for dimensionality reduction

- Principal component analysis (PCA)
- Autoencoder
- Feature selection

Principal Components Analysis

PCA

Principal components analysis (PCA)

- Principal components analysis (PCA) is a technique that can be used to simplify a dataset
- It is usually a linear transformation that chooses a new coordinate system for the data set such that
  - greatest variance by any projection of the dataset comes to lie on the first axis (then called the first principal component)
  - the second greatest variance on the second axis, and so on.
- PCA can be used for reducing dimensionality by eliminating the later principal components

Example

- Consider the following 3D points

1	2	4	3	5	6
2	4	8	6	10	12
3	6	12	9	15	18

- If each component is stored in a byte, we need 18 = 3 x 6 bytes

Example (cont.)

- Looking closer, we can see that all the points are related geometrically: they are all the same point, scaled by a factor:

$$\begin{bmatrix}1\\2\\3\end{bmatrix} = 1 \times \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}4\\8\\12\end{bmatrix} = 4 \times \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}5\\10\\15\end{bmatrix} = 5 \times \begin{bmatrix}1\\2\\3\end{bmatrix}$$

$$\begin{bmatrix}2\\4\\6\end{bmatrix} = 2 \times \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}3\\6\\9\end{bmatrix} = 3 \times \begin{bmatrix}1\\2\\3\end{bmatrix} \qquad \begin{bmatrix}6\\12\\18\end{bmatrix} = 6 \times \begin{bmatrix}1\\2\\3\end{bmatrix}$$

Example (cont.)

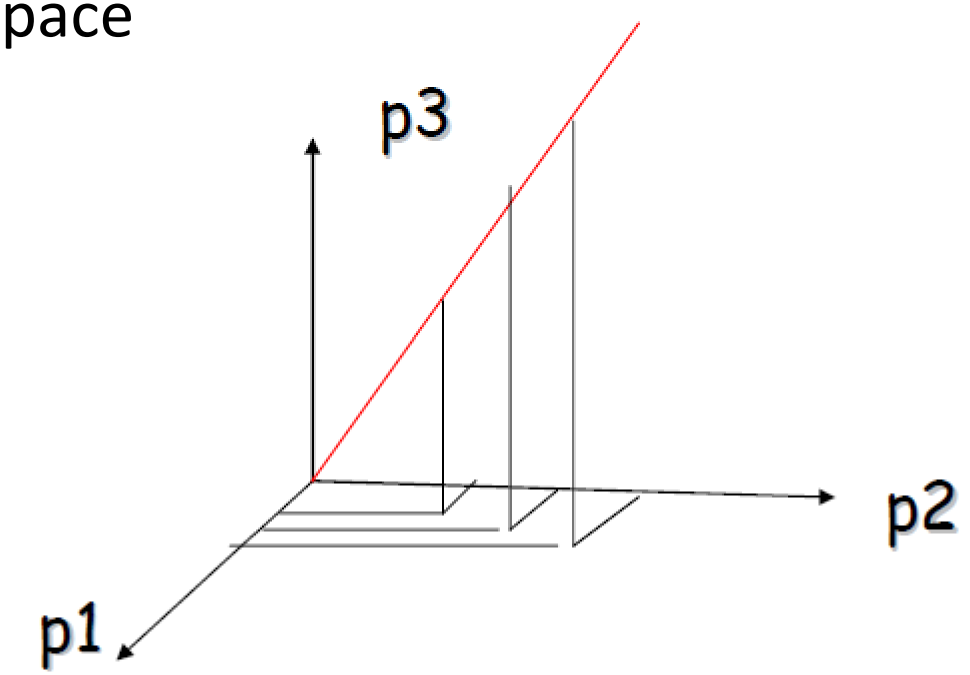
$$\begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} = 1 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} \qquad \begin{array}{|c|}\hline 4\\\hline 8\\\hline 12\\\hline\end{array} = 4 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} \qquad \begin{array}{|c|}\hline 5\\\hline 10\\\hline 15\\\hline\end{array} = 5 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}$$

$$\begin{array}{|c|}\hline 2\\\hline 4\\\hline 6\\\hline\end{array} = 2 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} \qquad \begin{array}{|c|}\hline 3\\\hline 6\\\hline 9\\\hline\end{array} = 3 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array} \qquad \begin{array}{|c|}\hline 6\\\hline 12\\\hline 18\\\hline\end{array} = 6 \times \begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}$$

- They can be stored using only 9 bytes (50% savings!):
  - Store one point (3 bytes) + the multiplying constants (6 bytes)

Geometrical interpretation

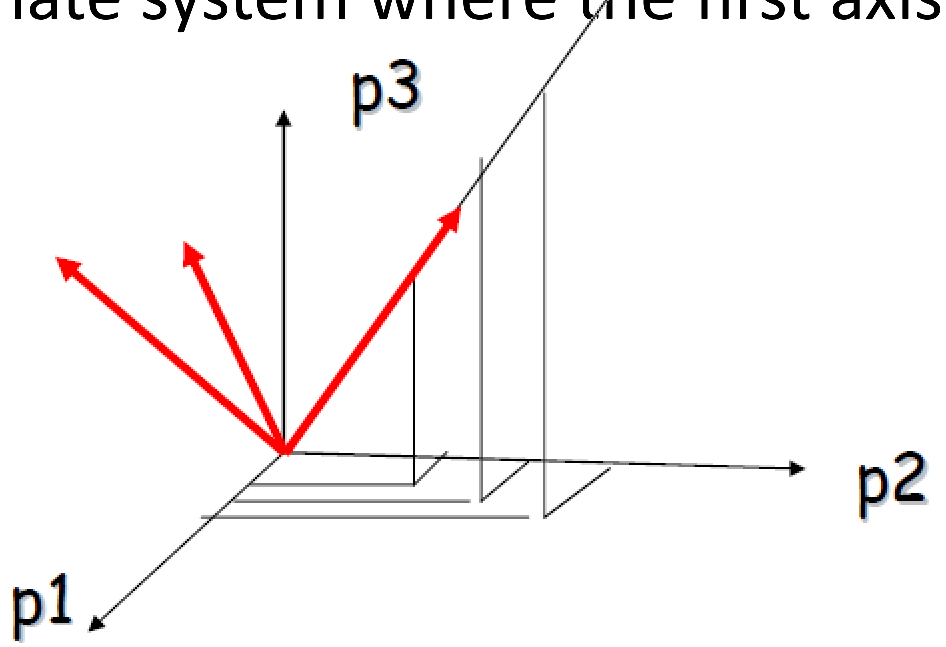
- View each point in 3D space



- In this example, all the points happen to belong to a line: a 1D subspace of the original 3D space

Geometrical interpretation

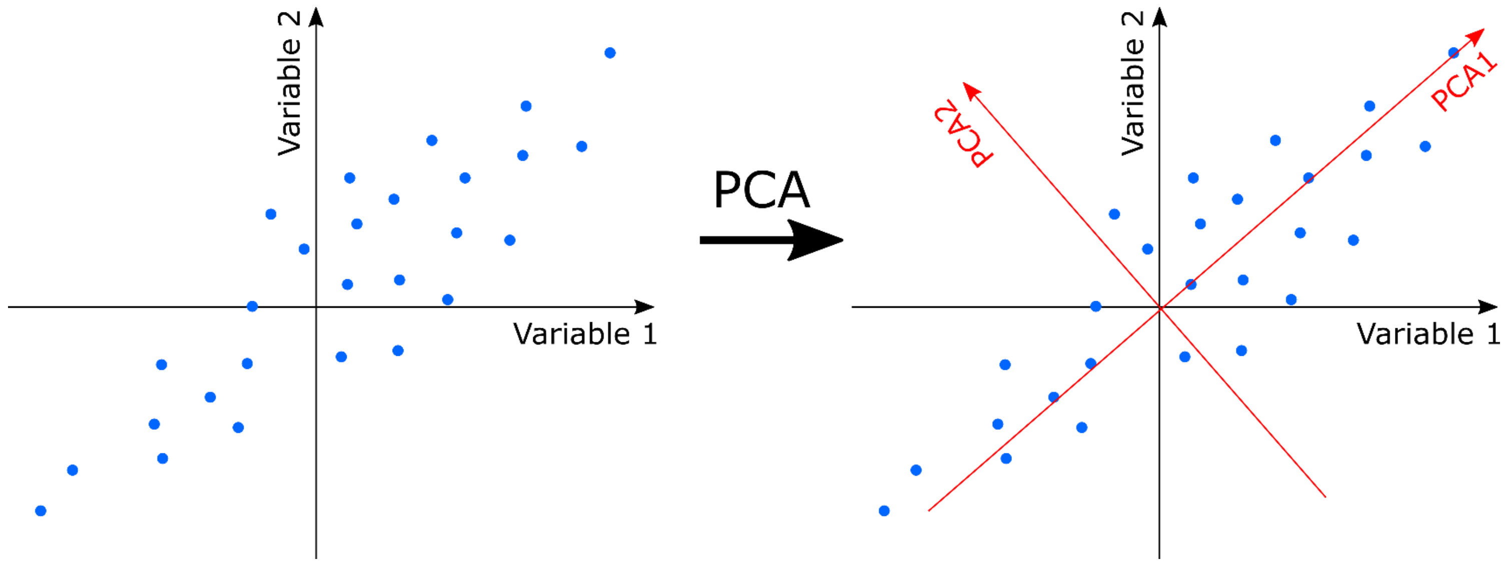
- Consider a new coordinate system where the first axis is along the direction of the line



- In the new coordinate system, every point has only one non-zero coordinate
  - we only need to store the direction of the line (a 3 bytes image) and the nonzero coordinate for each of the points (6 bytes)

Back to PCA

- Given a set of points, how can we know if they can be compressed like in the previous example?
  - The answer is to look into the correlation between the points
  - The tool for doing this is called PCA

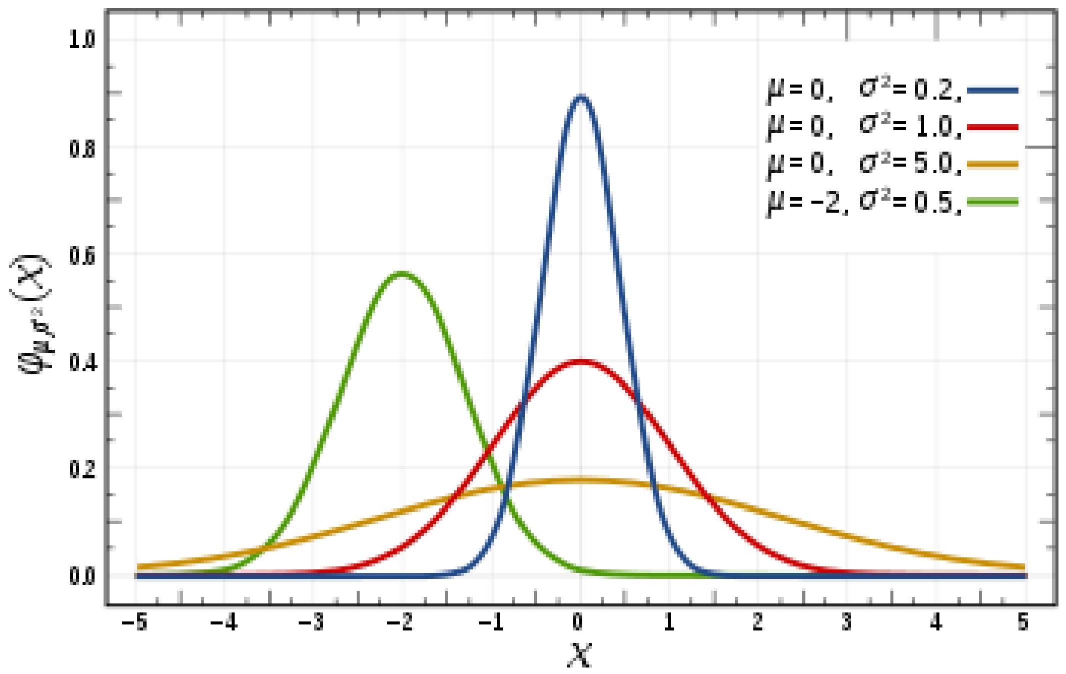


From example to theory

- In the example, we see that PCA rebuilds the coordination system for the data, by using
  - the direction with largest variance as the first new base direction
  - the direction with the second largest variance as the second new base direction
  - and so on
- The problem is, how can we find the direction with largest variance?
- The answer is the eigenvector for the covariance matrix of the data

Review – Variance

- Variance is the expectation of the squared deviation of a random variable from its mean
    - Informally, it measures how far a set of (random) numbers are spread out from their average value



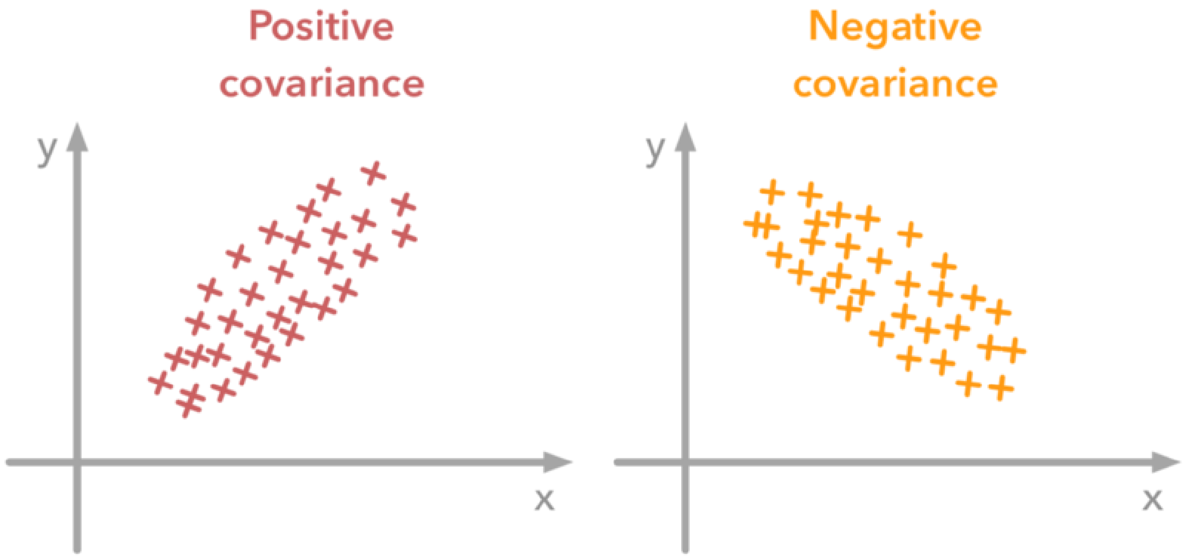
Review – Covariance

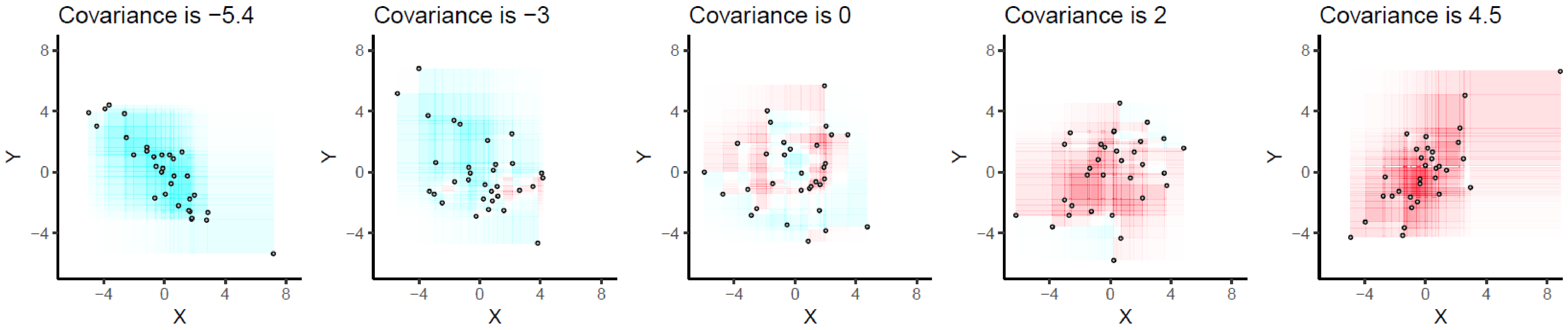
- Covariance is a measure of the joint variability of two random variables
  - If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the lesser values, (i.e., the variables tend to show similar behavior), the covariance is positive
    - E.g. as the number of hours studied increases, the marks in that subject increase
  - In the opposite case, when the greater values of one variable mainly correspond to the lesser values of the other, (i.e., the variables tend to show opposite behavior), the covariance is negative
  - The sign of the covariance therefore shows the tendency in the linear relationship between the variables
  - The magnitude of the covariance is not easy to interpret because it is not normalized and hence depends on the magnitudes of the variables. The normalized version of the covariance, the correlation coefficient, however, shows by its magnitude the strength of the linear relation

Review – Covariance (cont.)

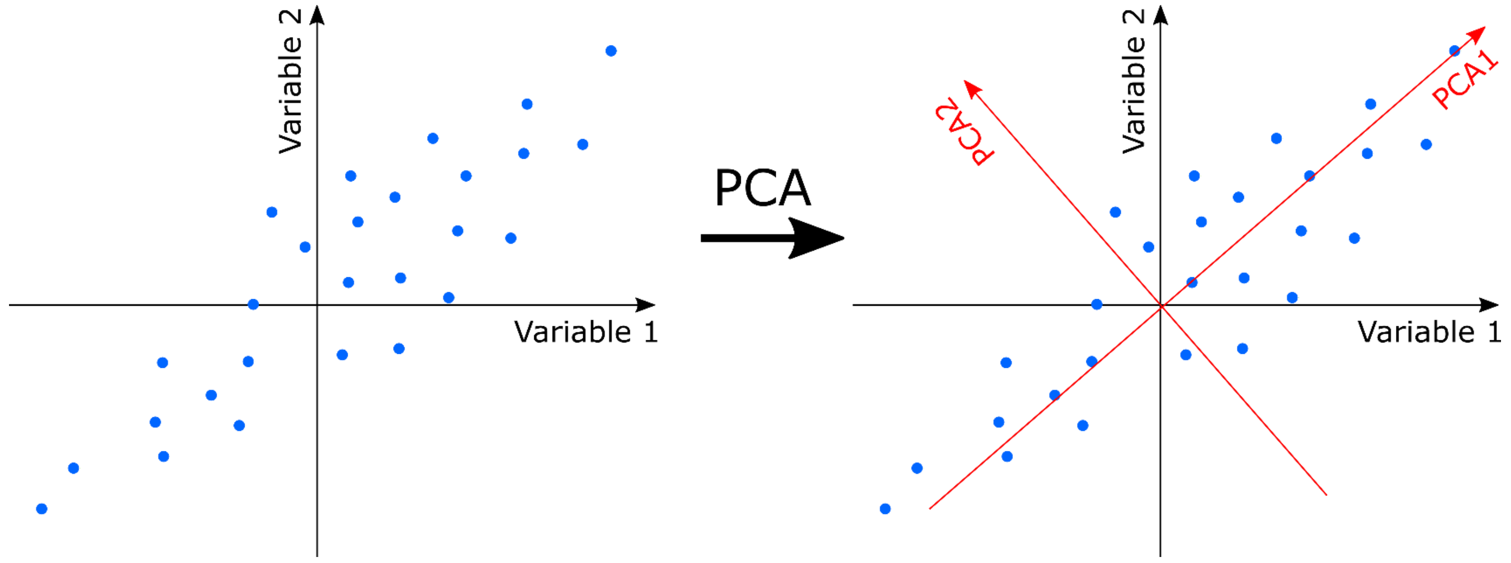
- Sample covariance

$$\text{covariance}(X,Y) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$





PCA



- PCA tries to identify the subspace in which the data approximately lies
- PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components
  - The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations

Covariance matrix

- Suppose there are 3 dimensions, denoted as X, Y, Z. The covariance matrix is

$$COV = \begin{bmatrix} COV(X,X) & COV(X,Y) & COV(X,Z) \\ COV(Y,X) & COV(Y,Y) & COV(Y,Z) \\ COV(Z,X) & COV(Z,Y) & COV(Z,Z) \end{bmatrix}$$

- Note the diagonal is the covariance of each dimension with respect to itself, which is just the variance of each random variable
- Also $COV(X,Y) = COV(Y,X)$
  - hence matrix is symmetric about the diagonal
- N-dimensional data will result in an $N \times N$ covariance matrix

Covariance in the covariance matrix

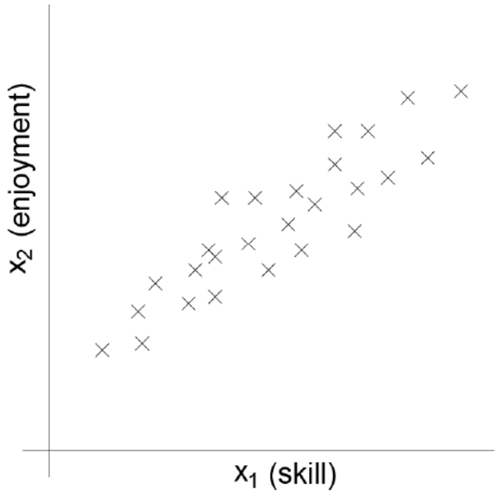
- Diagonal, or the variance, measures the deviation from the mean for data points in one dimension
- Covariance measures how one dimension random variable varies w.r.t. another, or if there is some linear relationship among them

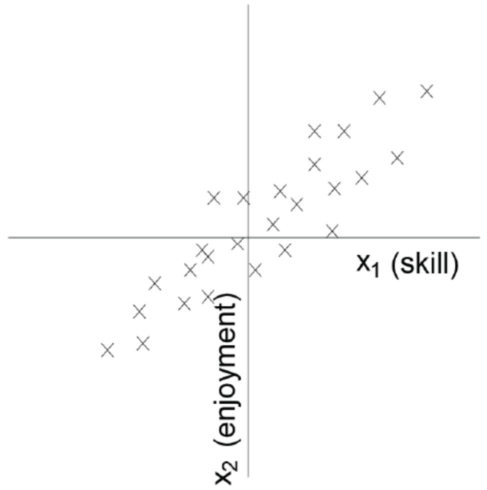
Data processing

- Given the dataset $D = \left\{x^{(i)}\right\}_{i=1}^{N}$
- Let $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$

$$X = \left[x^{(1)} - \bar{x} \quad x^{(2)} - \bar{x} \quad \dots \quad x^{(N)} - \bar{x}\right]$$

  - Move the center of the data set to 0



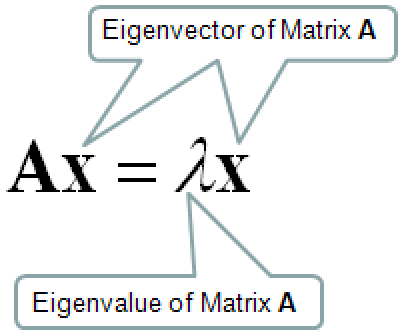


Data processing (cont.)

- $Q = XX^\top = \begin{bmatrix} x^{(1)} - \bar{x} & x^{(2)} - \bar{x} & \cdots & x^{(N)} - \bar{x} \end{bmatrix} \begin{bmatrix} \left(x^{(1)} - \bar{x}\right)^\top \\ \left(x^{(2)} - \bar{x}\right)^\top \\ \vdots \\ \left(x^{(N)} - \bar{x}\right)^\top \end{bmatrix}$
  - Q is square with d dimension
  - Q is symmetric
  - Q is the covariance matrix [aka scatter matrix]
  - Q can be very large (in vision, d is often the number of pixels in an image!)
    - For a 256×256 image, $d = 65536$!!
    - Don't want to explicitly compute Q

PCA

- By finding the eigenvalues and eigenvectors of the covariance matrix, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset
- This is the principal component
- PCA is a useful statistical technique that has found application in:
  - fields such as face recognition and image compression
  - finding patterns in data of high dimension


Eigenvector of Matrix **A**
$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$
Eigenvalue of Matrix **A**

PCA theorem

- Theorem: Each x_j can be written as: $x_j = \bar{x} + \sum_{i=1}^{d} g_{ji} e_i$ where e_i are the d eigenvectors of Q with non-zero eigenvalues

- **Notes:**
  1. The eigenvectors $e_1 e_2 \cdots e_d$ span an **eigenspace**
  2. $e_1 e_2 \cdots e_d$ are $d \times 1$ orthonormal vectors (directions in d-Dimensional space)
  3. The scalars g_{ji} are the coordinates of x_j in the space
$$g_{ji} = \langle x_j - \bar{x}, e_i \rangle$$

Using PCA to compress data

- Expressing x in terms of $e_1 e_2 \cdots e_d$ doesn't change the size of the data
- However, if the points are highly correlated, many of the new coordinates of x will become zero or closed to zero
- Sort the eigenvectors e_i according to their eigenvalue

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

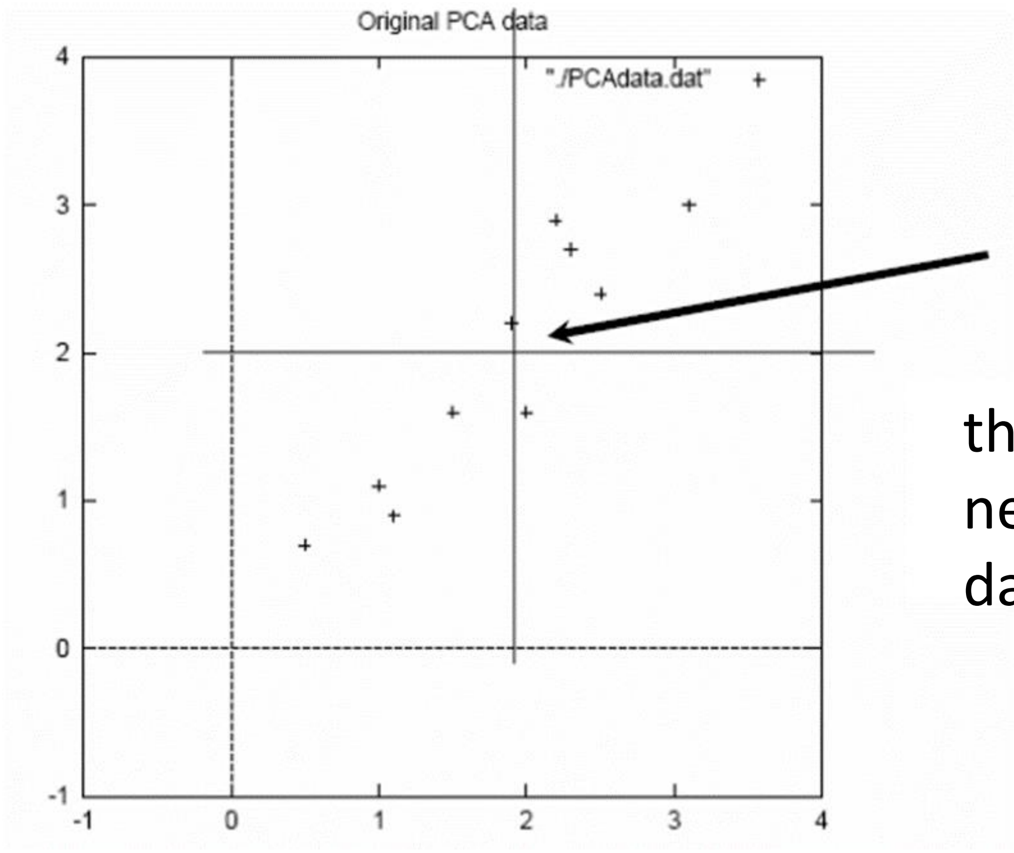
- Assume $\lambda_i \approx 0$ if $i > k$. Then

$$x_j \approx \bar{x} + \sum_{i=1}^{k} g_{ji} e_i$$

Example – STEP 1

DATA:

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



mean

this becomes the new origin of the data from now on

http://kybele.psych.cornell.edu/~edelman/Psych-465-Spring-2003/PCA-tutorial.pdf

Example – STEP 2

- Calculate the covariance matrix

$$\mathrm{Cov} = \begin{bmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{bmatrix}$$

- since the non-diagonal elements in this covariance matrix are positive, we should expect that both the x and y variable increase together

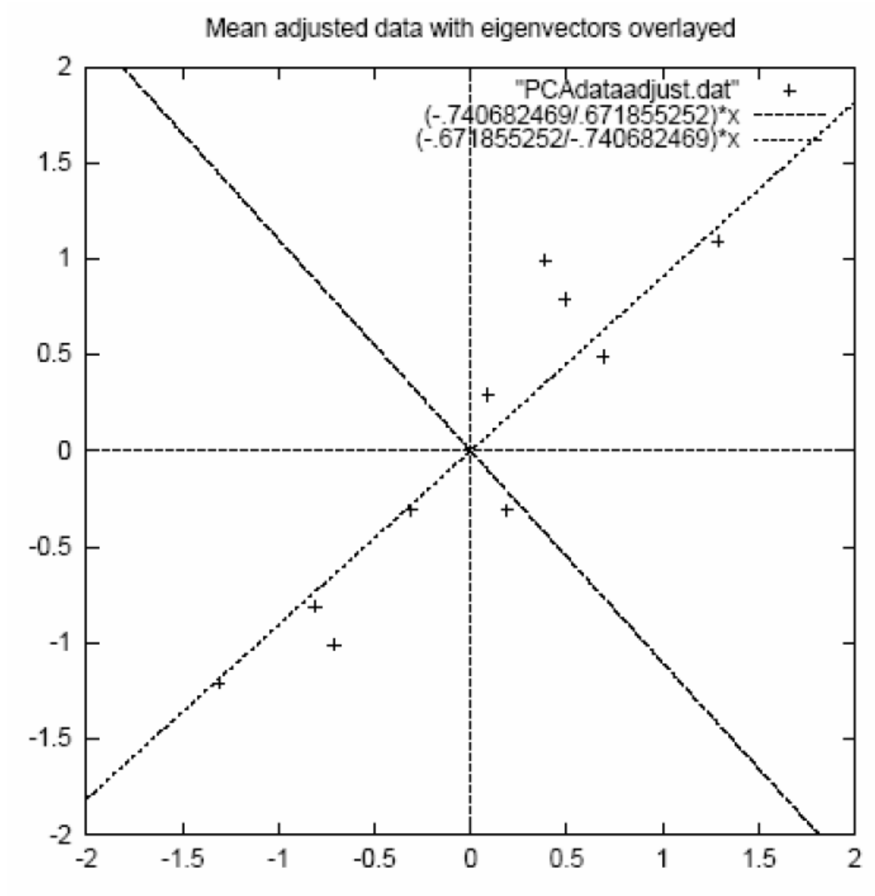
Example – STEP 3

- Calculate the eigenvectors and eigenvalues of the covariance matrix

- $\text{eigenvalues} = \begin{bmatrix} 0.0490833989 \\ 1.28402771 \end{bmatrix}$

- $\text{eigenvectors} = \begin{bmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{bmatrix}$

Example – STEP 3 (cont.)



- Eigenvectors are plotted as diagonal dotted lines on the plot
- Note they are perpendicular to each other
- Note one of the eigenvectors goes through the middle of the points, like drawing a line of best fit
- The second eigenvector gives us the other, less important, pattern in the data, that all the points follow the main line, but are off to the side of the main line by some amount

Example – STEP 4

- Feature Vector $= [e_1 \quad e_2 \quad \cdots \quad e_d]$

- We can either form a feature vector with both of the eigenvectors:

$$\begin{bmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{bmatrix}$$

- or, we can choose to leave out the smaller, less significant component and only have a single column:

$$\begin{bmatrix} -0.735178656 \\ 0.677873399 \end{bmatrix}$$

Example – STEP 5

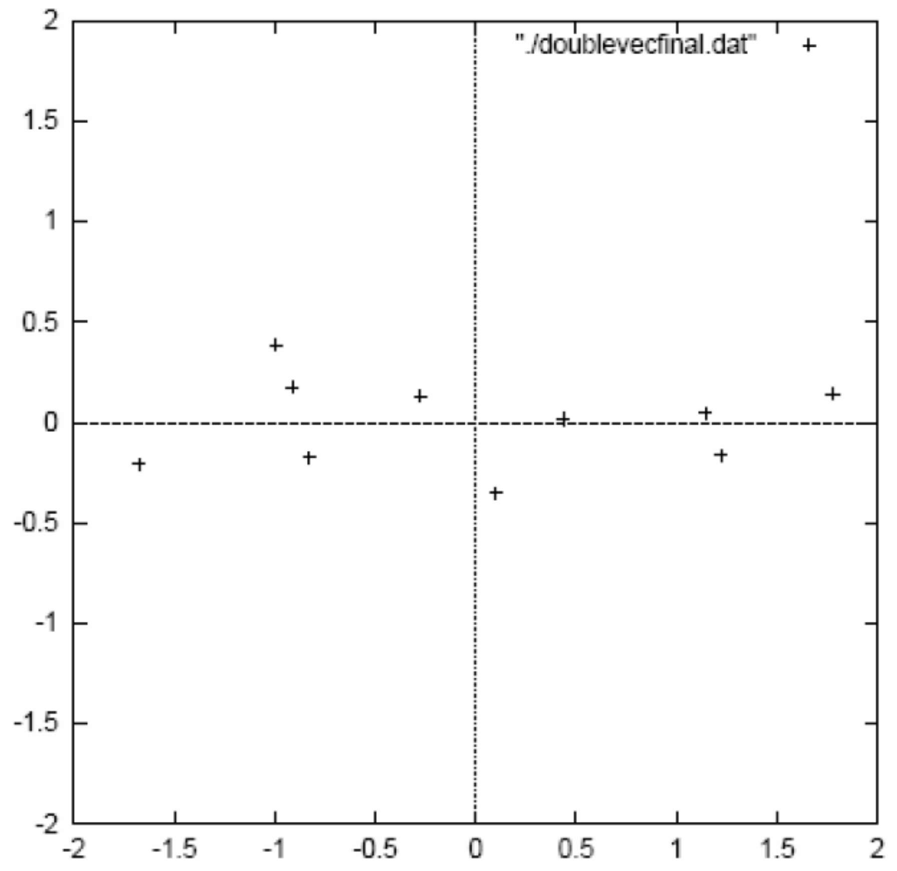
- Deriving new data coordinates

  FinalData = RowFeatureVector x RowZeroMeanData

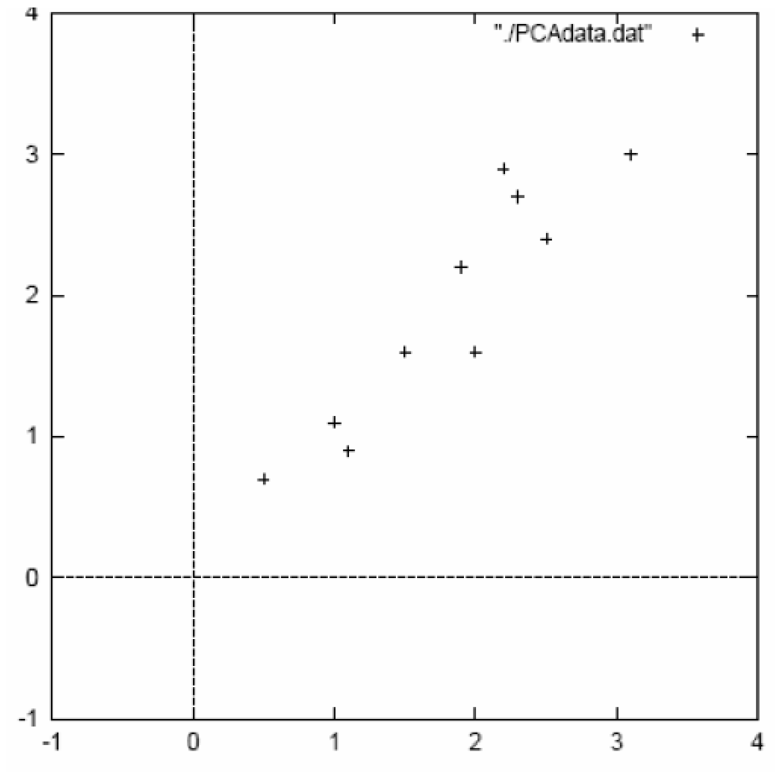
- RowFeatureVector is the matrix with the eigenvectors in the columns transposed so that the eigenvectors are now in the rows, with the most significant eigenvector at the top
- RowZeroMeanData is the mean-adjusted data transposed, ie. the data items are in each column, with each row holding a separate dimension
- Note: this is essentially rotating the coordinate axes so higher-variance axes come first

Example – STEP 5 (cont.)

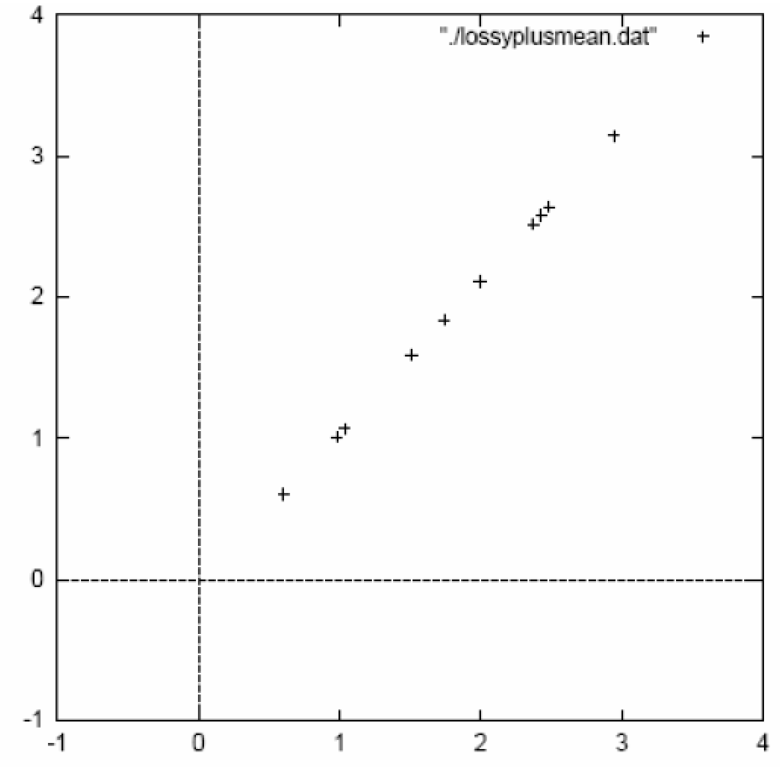
- The plot of the PCA results using both the two eigenvector



Example – Final approximation



2D point cloud

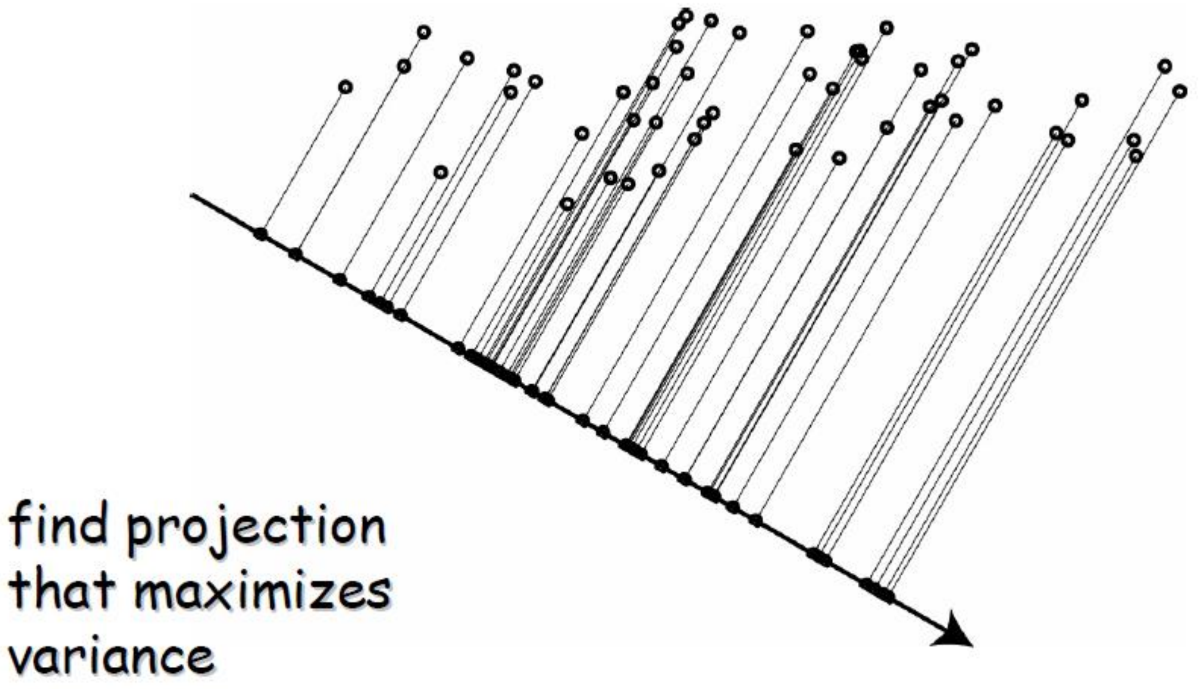


Approximation using one eigenvector basis

Revisit the eigenvectors in PCA

- It is critical to notice that the *direction of maximum variance* in the input space happens to be same as the *principal eigenvector of the covariance matrix*

- Why?



Revisit the eigenvectors in PCA (cont.)

- The projection of each point x to a direction u ($\|u\| = 1$) is
$$x^\top u$$
- The variance of the projection is
$$\frac{1}{N}\sum_{i=1}^{N}\left(\left(x^{(i)} - \bar{x}\right)^\top u\right)^2 = u^\top Q u$$
which is maximized when u is the eigenvector with the largest eigenvalue
- $Q = \sum_{i=1}^{d} \lambda_i e_i {e_i}^\top = E \Lambda E^\top$ with $\Lambda = \begin{bmatrix} \lambda_1 & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \lambda_N \end{bmatrix}$

Singular Value Decomposition

SVD

SVD

- Singular Value Decomposition (SVD) is a factorization method of matrix. It states that any $m \times n$ matrix A can be written as the product of 3 matrices:

$$A = U S V^T$$

- Where:
  - U is $m \times m$ and its columns are orthonormal eigenvectors of $AA^\top$
  - V is n$\times$ n and its columns are orthonormal eigenvectors of $A^\top A$
  - S is $m \times n$ is a diagonal matrix with r elements equal to the root of the positive eigenvalues of $AA^\top$ or $A^\top A$ (both matrices have the same positive eigenvalues anyway)

Visualization of SVD

$$A_{m\times n} = U_{m\times m} S_{m\times n} V^{T}_{n\times n}$$

$$\underset{m\times n}{\overset{\mathrm{A}}{\begin{pmatrix} x_{11} & x_{12} & & x_{1n} \\ & & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} = \underset{m\times m}{\overset{\mathrm{U}}{\begin{pmatrix} u_{11} & & u_{m1} \\ & \ddots & \\ u_{1m} & & u_{mm} \end{pmatrix}}} \underset{m\times n}{\overset{\mathrm{S}}{\begin{pmatrix} \sigma_1 & & & 0 \\ & \ddots & & \\ & & \sigma_r & \\ & & & \ddots \\ 0 & & & 0 \end{pmatrix}}} \underset{n\times n}{\overset{\mathrm{V^T}}{\begin{pmatrix} v_{11} & & v_{1n} \\ & \ddots & \\ v_{n1} & & v_{nn} \end{pmatrix}}}$$

Example of SVD

- Let's assume：

$$A = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix}$$

- We can have:

$$AA^T = \begin{pmatrix} 17 & 8 \\ 8 & 17 \end{pmatrix} \qquad A^T A = \begin{pmatrix} 13 & 12 & 2 \\ 12 & 13 & -2 \\ 2 & -2 & 8 \end{pmatrix}$$

Example of SVD

- Compute U and V respectively:

$$AA^T = \begin{pmatrix} 17 & 8 \\ 8 & 17 \end{pmatrix}$$

eigenvalues: $\lambda_1 = 25$, $\lambda_2 = 9$

eigenvectors

$$u_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \quad u_2 = \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$$

$$A^TA = \begin{pmatrix} 13 & 12 & 2 \\ 12 & 13 & -2 \\ 2 & -2 & 8 \end{pmatrix}$$

eigenvalues: $\lambda_1 = 25$, $\lambda_2 = 9$, $\lambda_3 = 0$

eigenvectors

$$v_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{pmatrix} \quad v_2 = \begin{pmatrix} 1/\sqrt{18} \\ -1/\sqrt{18} \\ 4/\sqrt{18} \end{pmatrix} \quad v_3 = \begin{pmatrix} 2/3 \\ -2/3 \\ -1/3 \end{pmatrix}$$

Example of SVD

- Finally, we have:

$$A = USV^T = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{pmatrix}$$

Application

- Matrix factorization in recommendation system

- Suppose in the database of Taobao, there are m items and n users, and a $m \times n$ binary matrix A that indicates whether a user has or has not bought an item. As m is usually very large and users only buy very limited number of items, the matrix is very sparse
- As the manager of Taobao, how can you predict the likelihood that a user will buy a specific item?
- SVD is the suitable tool to help the Taobao manager

Autoencoder

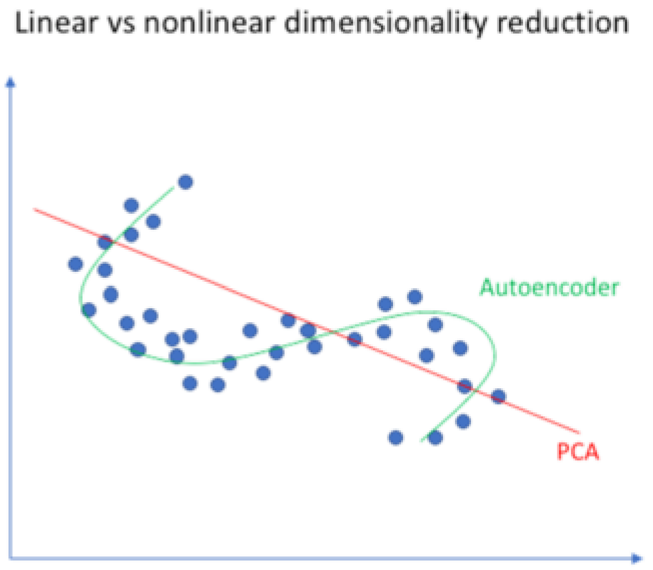
https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoder

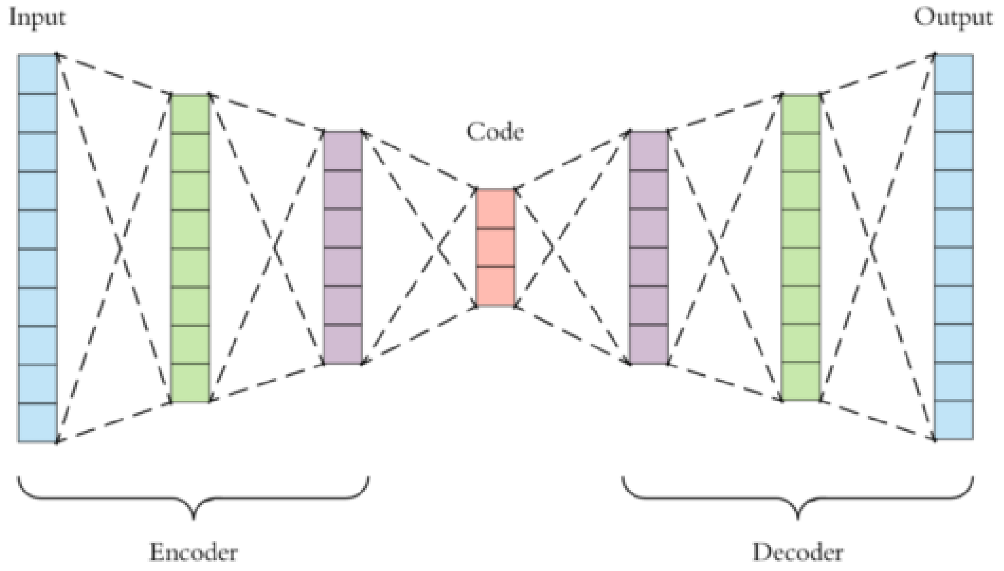
- An autoencoder **neural network** is an **Unsupervised Machine learning** model that applies backpropagation, setting the target values to be equal to the inputs.
- Autoencoders are used to reduce the size of our inputs into a smaller representation.
- If anyone needs the original data, they can reconstruct it from the compressed data.

Autoencoder V.S. PCA

- Autoencoders are preferred over PCA because:
  - An autoencoder can learn non-linear transformations with a non-linear activation function and multiple layers.
  - It doesn't have to learn dense layers. It can use convolutional layers to learn which is better for video, image and series data.
  - It is more efficient to learn several layers with an autoencoder rather than learn one huge transformation with PCA.
  - An autoencoder provides a representation of each layer as the output.
  - It can make use of pre-trained layers from another model to apply transfer learning to enhance the encoder/decoder.



Architecture of Autoencoders



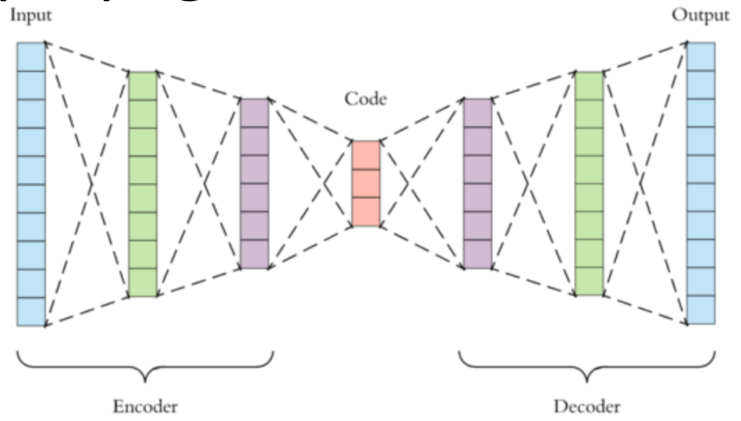
- **Encoder**: This part of the network compresses the input into a latent space representation. The encoder layer encodes the input image as a compressed representation in a reduced dimension. The compressed image is the distorted version of the original image.
- **Code**: This part of the network represents the compressed input which is fed to the decoder.
- **Decoder**: This layer decodes the encoded image back to the original dimension. The decoded image is a lossy reconstruction of the original image and it is reconstructed from the latent space representation.

Training the Autoencoder

- Let f be the function of the encoder, ie, $f(input) = code$
- Let g be the function of the decoder, ie, $g(code) = output$
- Autoencoder is trying approximate the input using the output, or trying to find f and g that minimize the following loss:

$$J = \sum (output \ - input)^2$$

- The above loss can be minimized using backpropagation

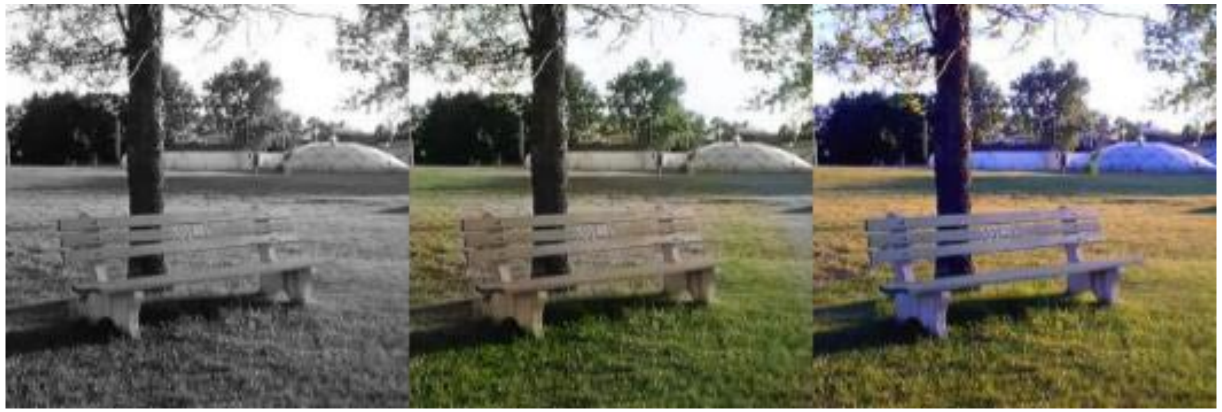


Hyperparameters

- **Code size**: It represents the number of nodes in the middle layer. Smaller size results in more compression.
- **Number of layers**: The autoencoder can consist of as many layers as we want.
- **Number of nodes per layer**: The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. The decoder is symmetric to the encoder in terms of the layer structure.
- **Loss function**: We either use mean squared error or binary cross-entropy. If the input values are in the range [0, 1] then we typically use cross-entropy, otherwise, we use the mean squared error.

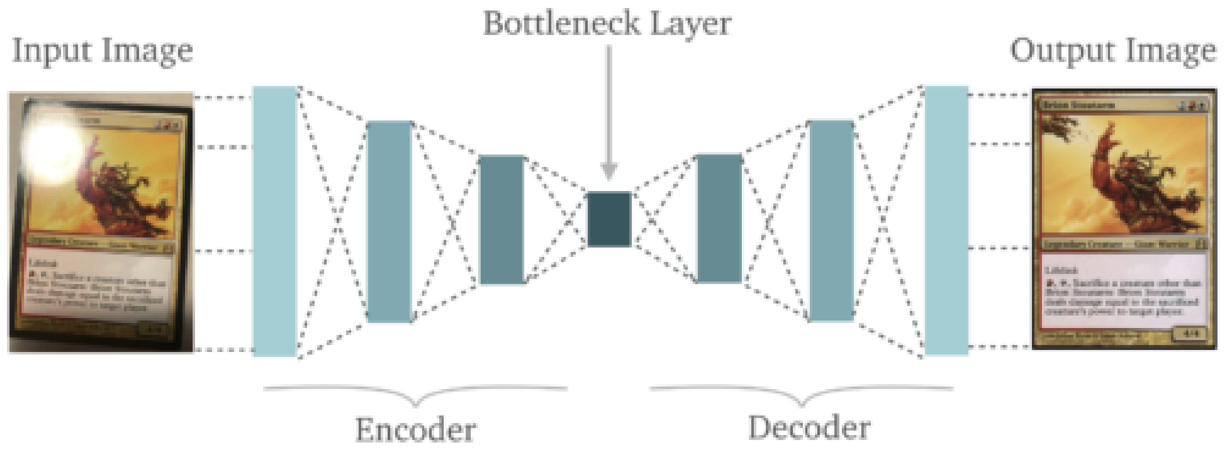
Application-Image Coloring

- Autoencoders are used for converting any black and white picture into a colored image. Depending on what is in the picture, it is possible to tell what the color should be.



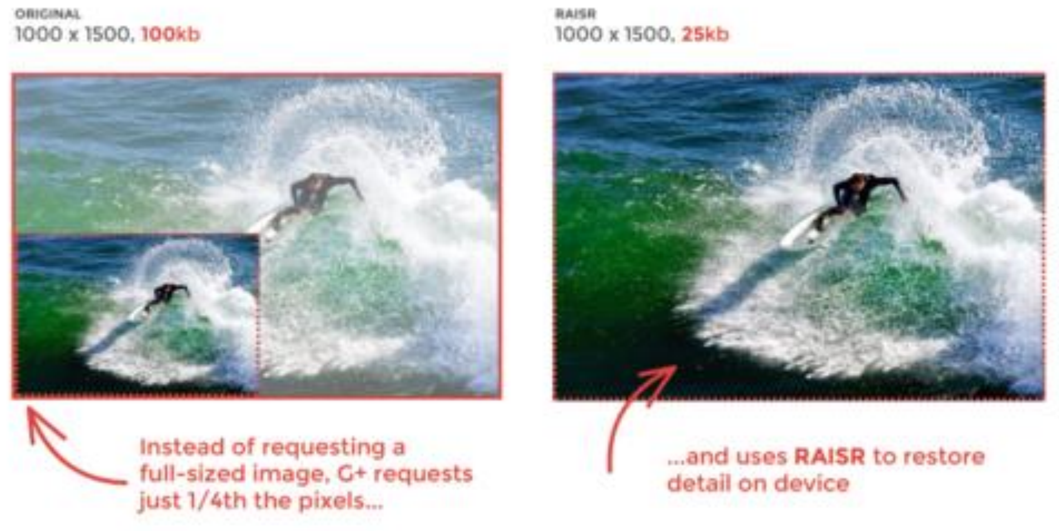
Application-Feature variation

- It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.



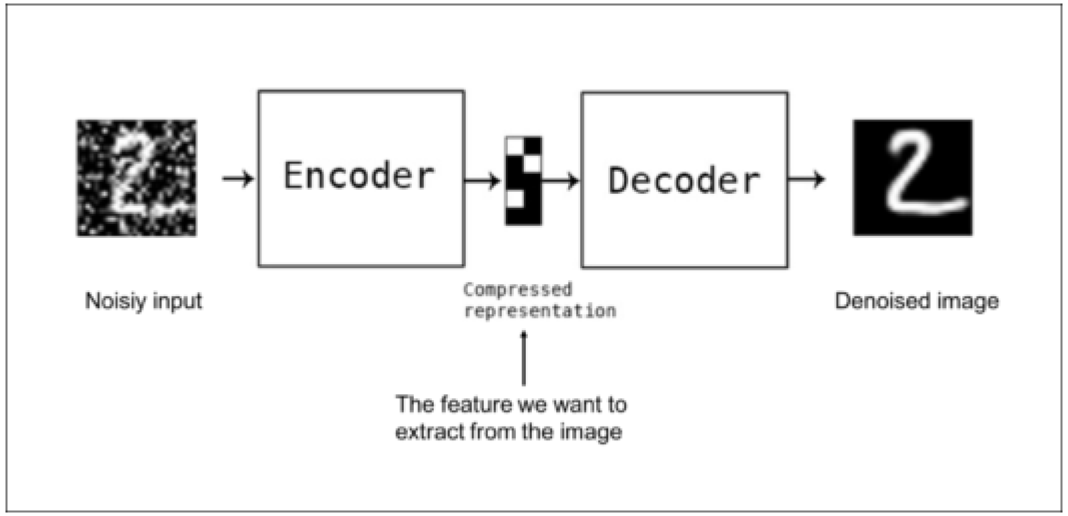
Application-Dimensionality Reduction

- The reconstructed image is the same as our input but with reduced dimensions. It helps in providing the similar image with a reduced pixel value.


ORIGINAL
1000 x 1500, 100kb
RAISR
1000 x 1500, 25kb
Instead of requesting a full-sized image, G+ requests just 1/4th the pixels...
...and uses RAISR to restore detail on device

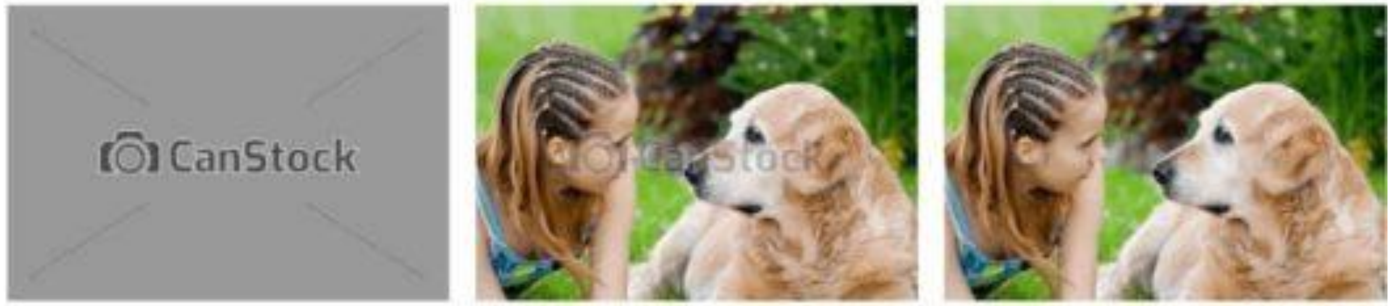
Application – Denoising image

- The input seen by the autoencoder is not the raw input but a stochastically corrupted version. A denoising autoencoder is thus trained to reconstruct the original input from the noisy version.


Encoder
Decoder
Noisiy input
Compressed representation
Denoised image
The feature we want to extract from the image

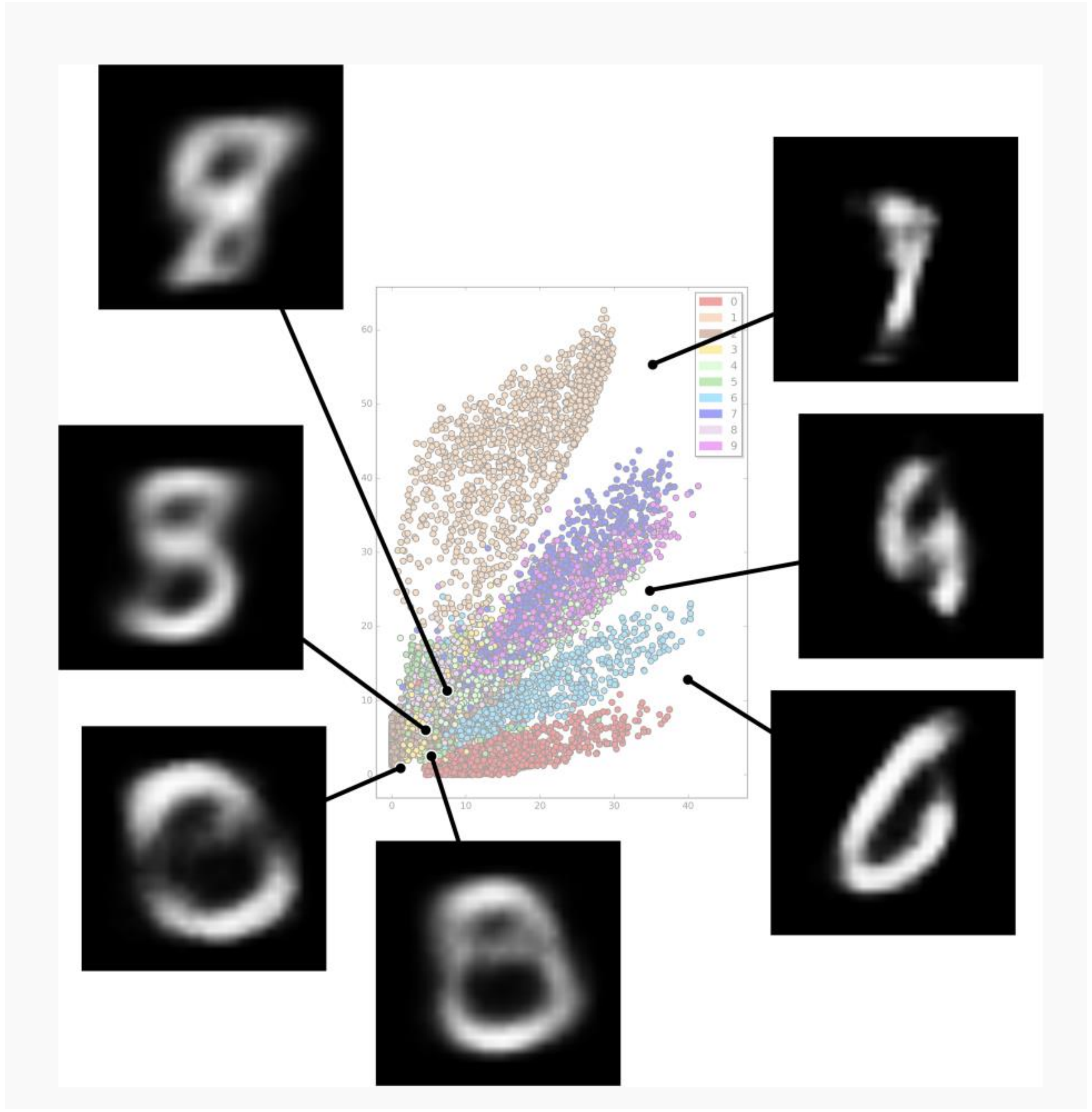
Application-Watermark Removal

- It is also used for removing watermarks from images or to remove any object while filming a video or a movie.



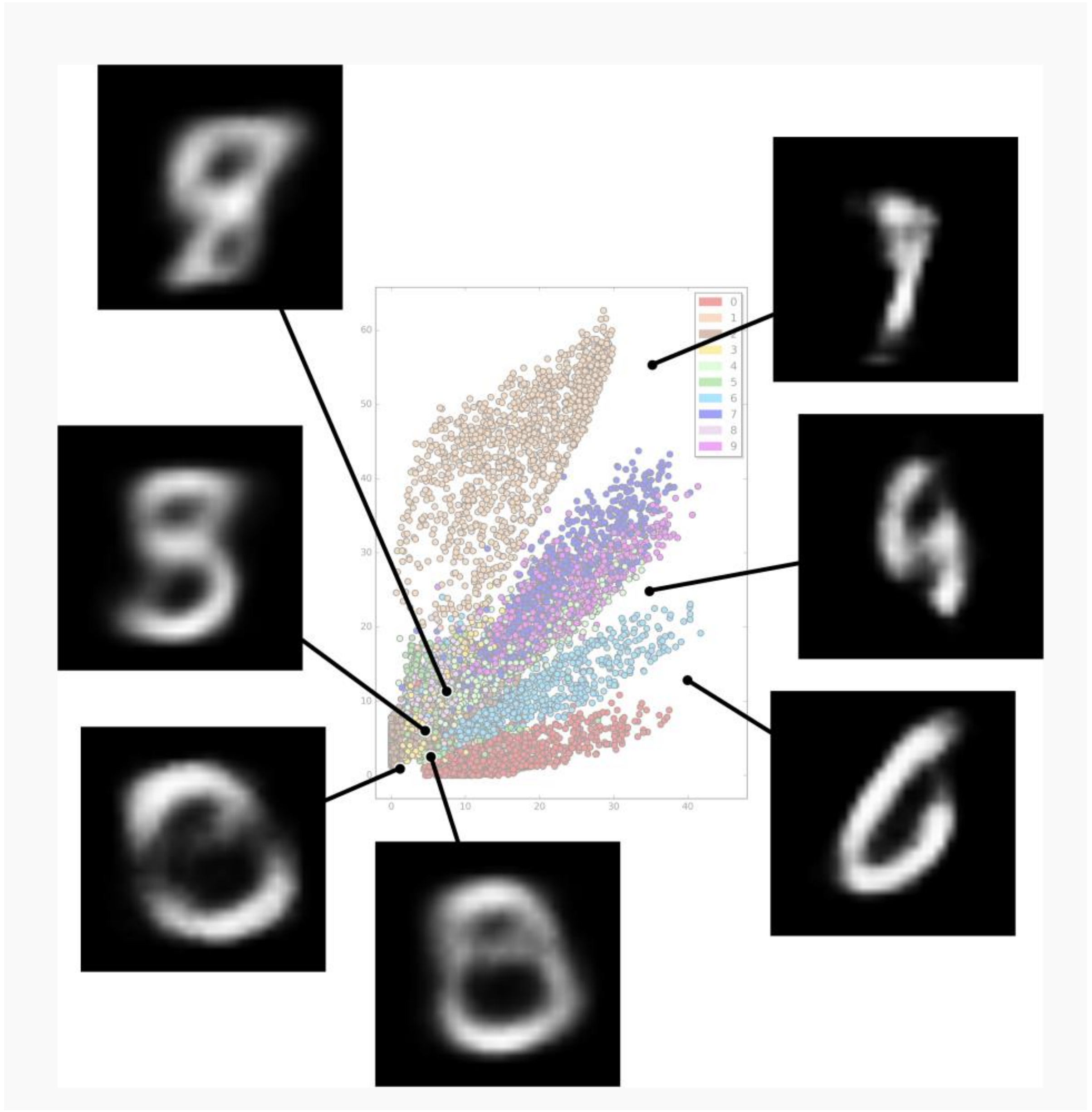
Challenges

- Gaps in the latent space
- Separability in the latent space
- Discrete latent space



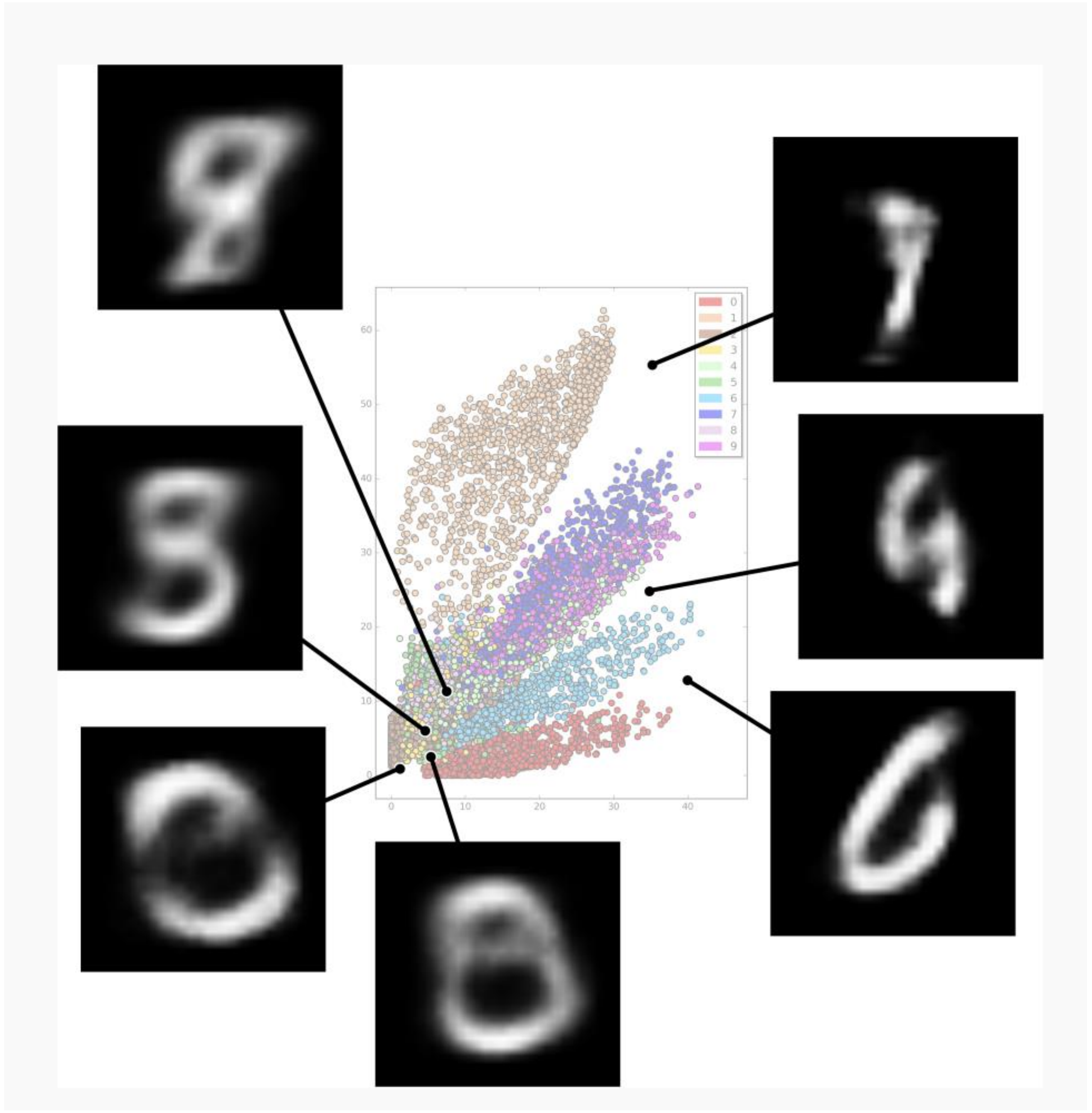
Challenges example

- We can see that the latent space contains gaps, and we do not know what characters in these spaces may look like. This is equivalent to having a lack of data in a supervised learning problem, as our network has not been trained for these circumstances of the latent space.



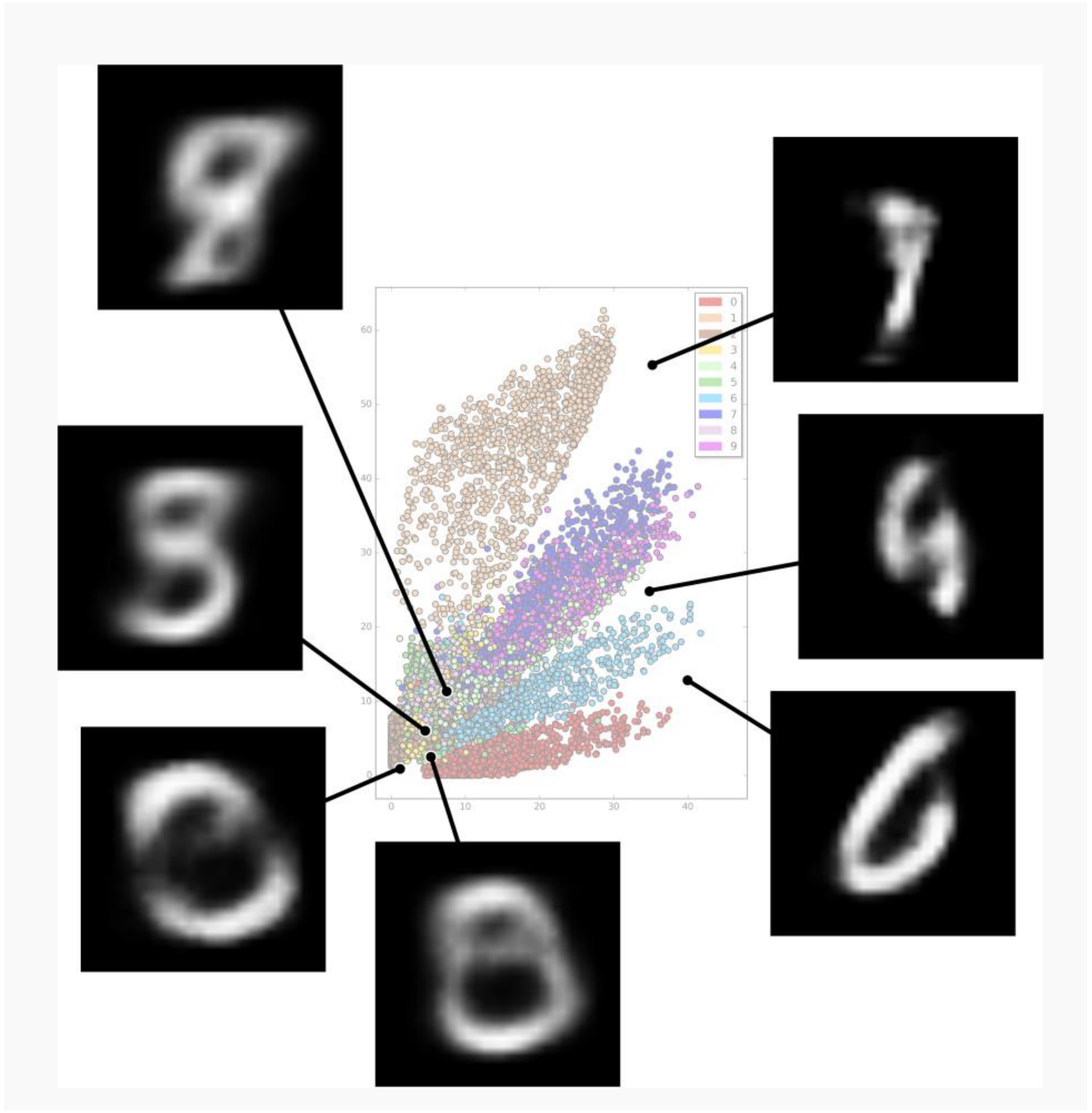
Challenges example

- Another issue is the separability of the spaces, several of the numbers are well separated in the above figure, but there are also regions where the labeled is randomly interspersed, making it difficult to separate the unique features of characters (in this case the numbers 0–9).



Challenges example

- Another issue here is the inability to study a continuous latent space, for example, we do not have a statistical model that has been trained for arbitrary input (and would not even if we closed all gaps in the latent space).



Feature Selection

Feature selection

- The most important problem with PCA and Autoencoder is that the new feature is some combination of the existing feature, which is hard to interpret.

- In the example of predicting the health condition of some students, we have the following feature:
  - Weight in kilogram.
  - Height in inch, height in cm
  - Hours of sport per day
  - Favorite color
  - Scores in math

Feature selection

- In the example
  - The new feature produced by PCA could be some thing like $0.3 * height + 0.62 * weight$
  - The new feature produced by Autoencoder could be some thing like $(0.1 * height + weight/height)^2$
- All of these new features are hard to interpret.
- If we only want to *select* a subset of feature, instead of *transform* the original feature into a smaller set, we need feature selection algorithm.

Feature selection

- *Feature Selection* is a process that chooses an optimal subset of features according to a certain criterion.

- Why we need FS:
  1. to improve performance (in terms of speed, predictive power, simplicity of the model).
  2. to visualize the data for model selection.
  3. To reduce dimensionality and remove noise.

Feature types

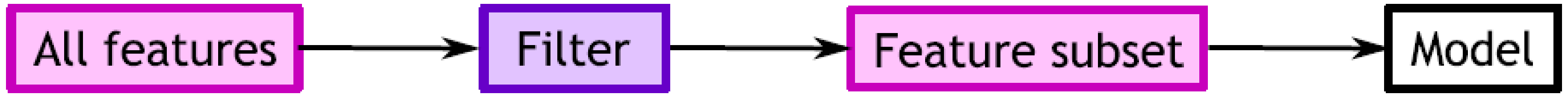
- **Relevant features** - those that we need to perform well
- **Irrelevant features** - those that are simply unnecessary
- **Redundant features** - those that become irrelevant in the presence of others
- Feature selection works by finding the relevant feature and illuminating the irrelevant feature and redundant feature

Feature selection algorithms

- Feature Selection algorithms can be classified into 3 main categories:
    - Wrappers
    - Filters
    - Embedded methods

Filter

- **Filter**: selects a subset of variables independently of the model that shall use them.



- It is a one-shot process (not iterative).
- It provides a set of "the most important" variables as the resulting subset *independently of the employed model.*

Filter Example-mRMR

- Minimum-redundancy-maximum-relevance (mRMR) algorithms tries to find a sub set of features with the maximum relevance to the label while minimize the redundancy within the features.

- The relevance of a feature set *S* for the class *c* is defined by the average value of all mutual information values between the individual feature f_i and the class *c* as follows

$$D(S, c) = \frac{1}{|S|} \sum_{f_i \in S} I(f_i; c).$$

Filter Example-mRMR

- The redundancy of all features in the set *S* is the average value of all mutual information values between the feature f_i and the feature f_j:

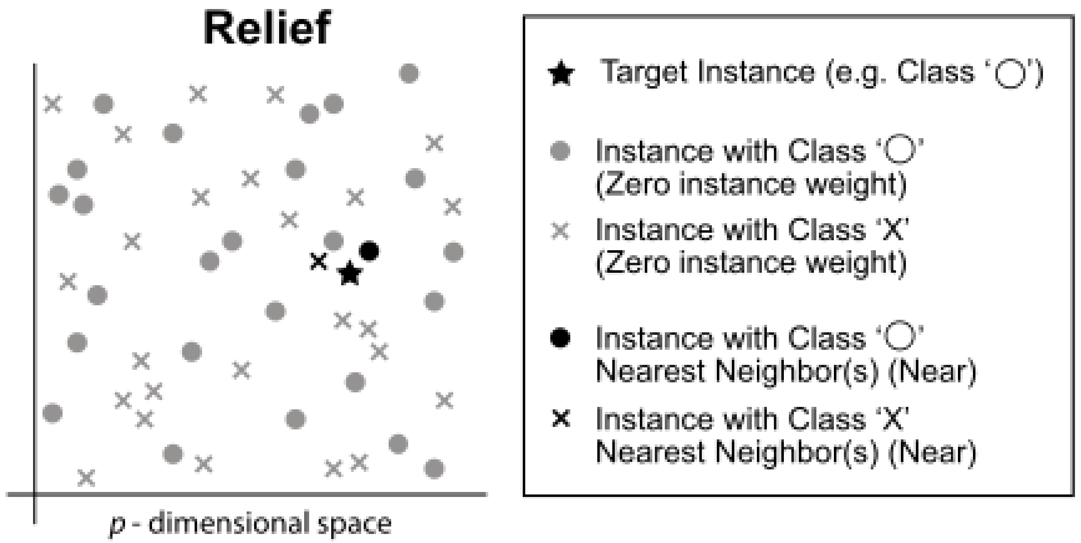
$$R(S) = \frac{1}{|S|^2} \sum_{f_i, f_j \in S} I(f_i; f_j)$$

- The mRMR criterion is a combination of two measures given above and is defined as follows:

$$\mathrm{mRMR} = \max_S \left[\frac{1}{|S|} \sum_{f_i \in S} I(f_i; c) - \frac{1}{|S|^2} \sum_{f_i, f_j \in S} I(f_i; f_j) \right]$$

Filter Example-Relief

- Relief feature scoring is based on the identification of feature value differences between nearest neighbor instance pairs.
  - If a feature value difference is observed in a neighboring instance pair with the same class (a 'hit'), the feature score decreases.
  - If a feature value difference is observed in a neighboring instance pair with different class(a 'miss'), the feature score increases.

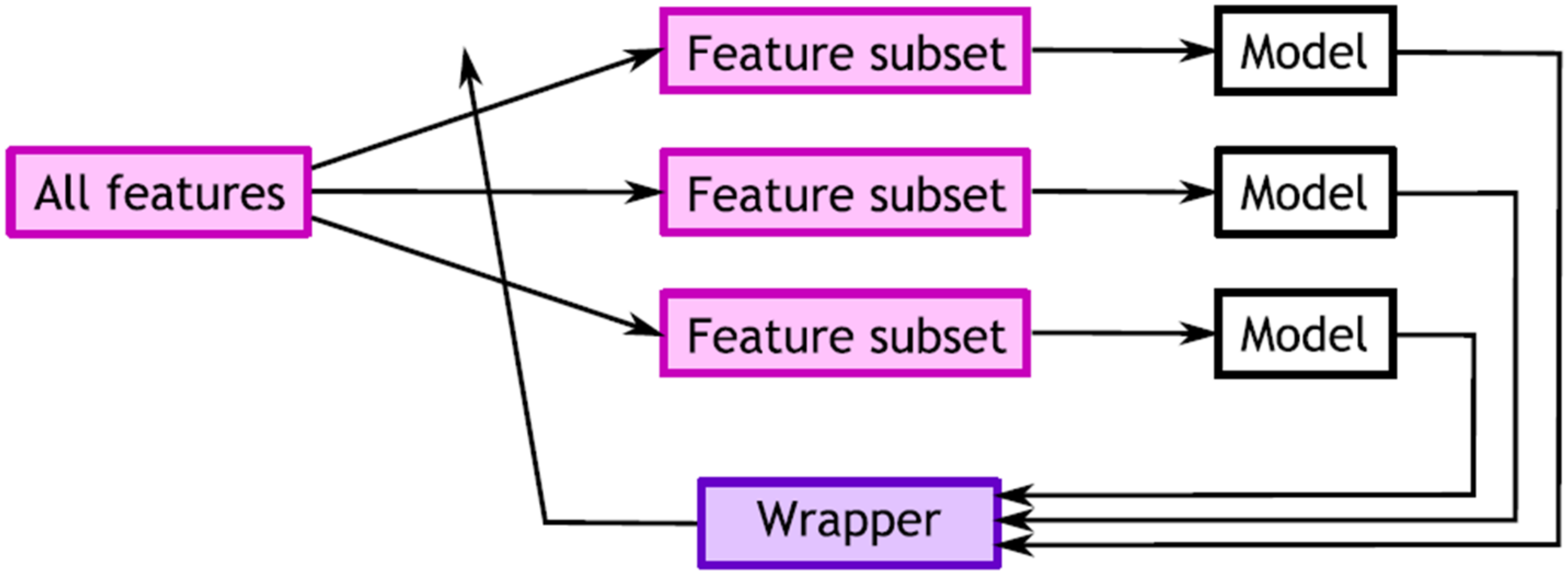

Relief
p - dimensional space
★ Target Instance (e.g. Class '○')
● Instance with Class '○' (Zero instance weight)
× Instance with Class 'X' (Zero instance weight)
● Instance with Class '○' Nearest Neighbor(s) (Near)
× Instance with Class 'X' Nearest Neighbor(s) (Near)

Filter Example-Relief

- The general workflow of Relief algorithm
- For each feature f_i, initialized its score w_i as 0.
- For each feature f_i
  - For each sample s_j, find its nearest hit h and nearest miss m
    - if s_j and h have different value at f_i, w_i decrease by 1.
    - if s_j and m have different value at f_i, w_i increase by 1.
- Return the top k features with highest score.

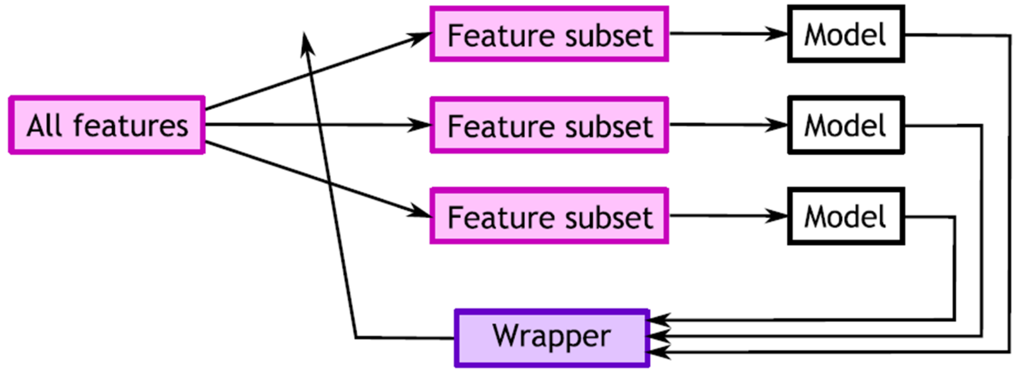
Wrapper

- **Wrapper:** selects a subset of variables taking into account the model that will use them.



Wrapper

- **Wrapper:** selects a subset of variables taking into account the model that will use them.



  - It is an iterative process.
  - In each iteration, several subsets of input variables is generated and tested on the particular model type.
  - According to the success of the model for individual feature subsets, it is chosen which subsets will be tested in the next iteration.
  - Feature selection is here part of the model training; we need separate testing data to evaluate the final model error.

Wrapper Example-SFS

- Sequential Forward Selection (SFS) is a greedy search algorithm that attempts to find the “optimal” feature subset by iteratively selecting features based on the classifier performance.
  - Start with an empty set S, represent all features as set F.
  - Select the feature f from F, which can help the model achieve the highest score on S+f.
  - S = S+f, F = F-f.
  - Repeat the steps util there are k features in S.

Embedded

- The embedded algorithm integrated the feature selection process into other algorithms.

- A famous example is the L1-regularization, or LASSO algorithm.