University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 11 - Kernel Methods
The Kernel Trick - Support Vector Machines - Inequality Constraint Optimization - Kernel SVM + Quick revisit of PCA

Erik Bekkers

Catch-up:
Probabilistic PCA
Non-linear PCA

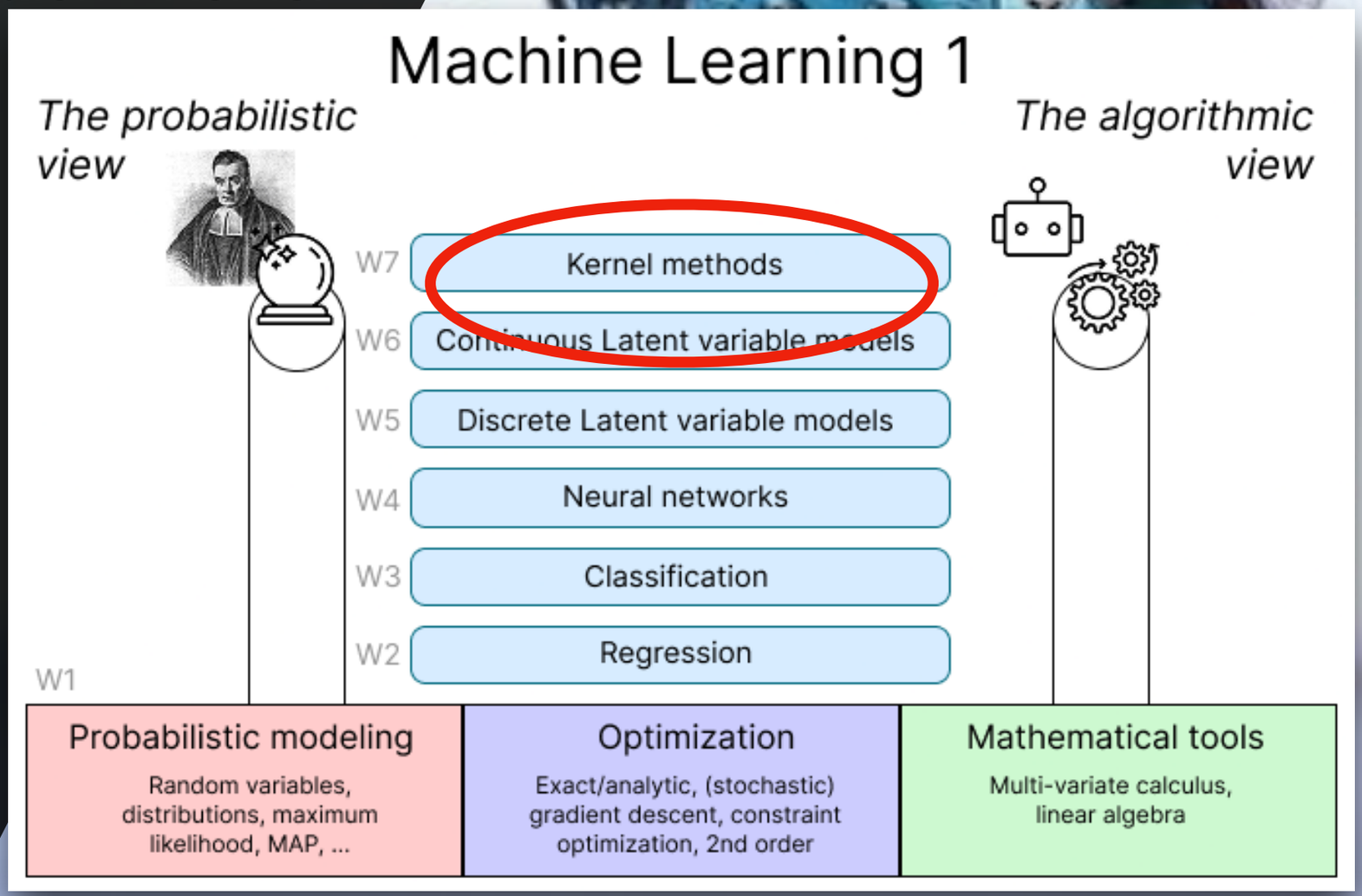


Image credit: Kirillm | Getty Images

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 10 - Unsupervised Learning

Continuous latent variable models

Erik Bekkers

Slide credits: Thomas Kipf and Giorgio Patrini

Image credit: Kirillm | Getty Images

Advanced generative modeling (sep 2022): “Stable Diffusion”

Based on*

- Advanced Probabilistic modeling (non-linear continuous latent variable models)
- Deep Learning
- Natural Language Processing

*Relevant UVA Courses:
Machine Learning 1 (and 2), Natural Language Processing 1 (and 2), Deep Learning 1 (and 2), Interpretability and Explainability in AI, Fairness, Accountability, Confidentiality and Transparency in AI

Prompt: portrait photo of a asia old warrior chief, tribal panther make up, blue on red, side profile, looking away, serious eyes, 50mm portrait photography, hard rim lighting photography–beta –ar 2:3 –beta –upbeta –upbeta

Prompt: beautiful open kitchen in the style of elena of avalor overlooking aerial wide angle view of a solarpunk vibrant city with greenery, interior architecture, kitchen, eating space, rendered in octane, in the style of Luc Schuiten, craig mullins, solarpunk in deviantart, photorealistic, highly detailed, Vincent Callebaut, elena of avalor, highly detailed, –ar 16:9

Prompt: infinite hyperbolic intricate maze, futuristic eco warehouse made out of dead vines, glass mezzanine level, lots of windows, wood pallets, designed by Aesop, forest house surrounded by massive willow trees and vines, white exterior facade, in full frame, , exterior view, twisted house, 3d printed canopy, clay, earth architecture, cavelike interiors, convoluted spaces, hyper realistic, photorealism, octane render, unreal engine, 4k, –stylize 5000 –ar 1:2

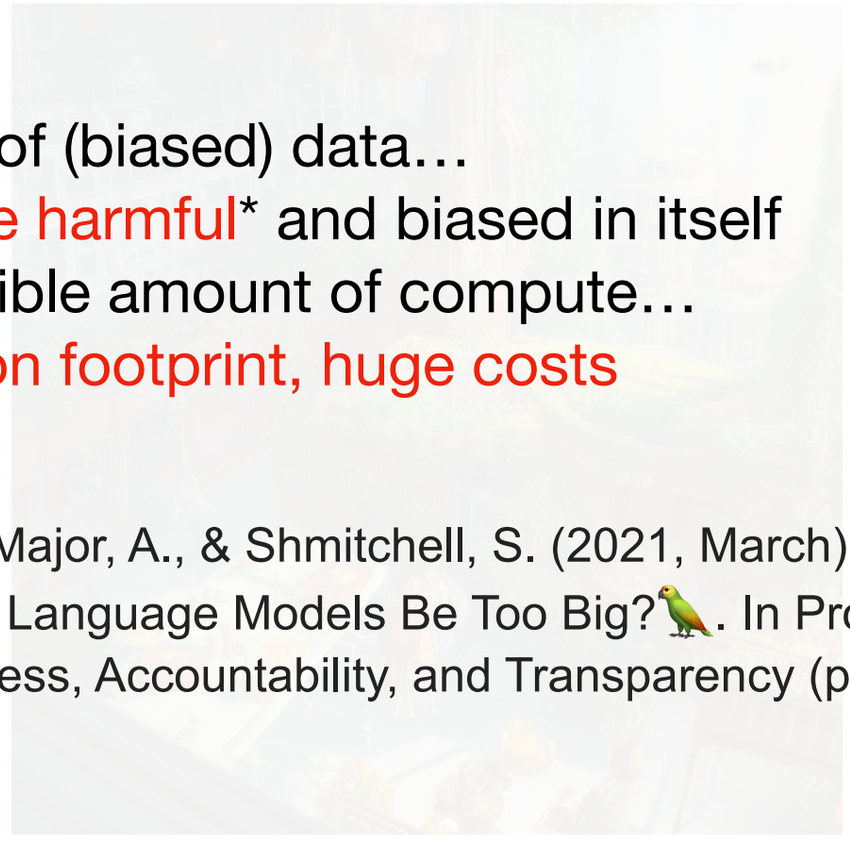
source: https://mpost.io/best-100-stable-diffusion-prompts-the-most-beautiful-ai-text-to-image-prompts/

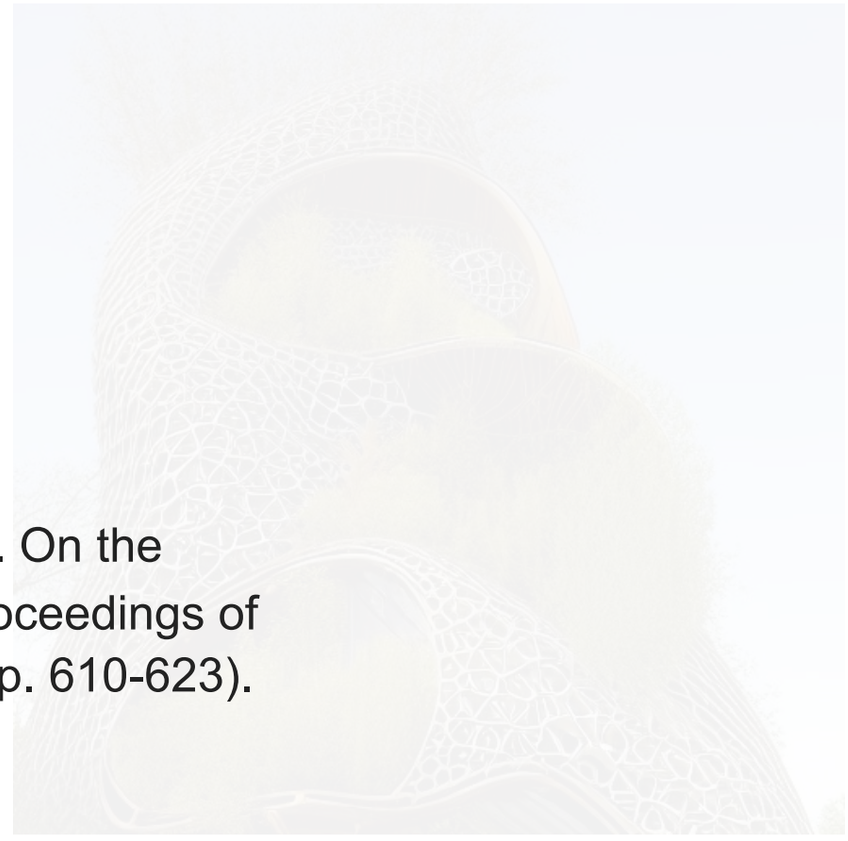
Advanced generative modeling (sep 2022): “Stable Diffusion”

Based on
- Advanced Probabilistic modeling (non-linear continuous latent variable models)
- Deep Learning
- Natural Language Processing

Exciting, but…
- Requires tons of (biased) data…
  - Data can be harmful* and biased in itself
- Require incredible amount of compute…
  - Huge carbon footprint, huge costs

Recommended read:
Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021, March). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?🦜. In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (pp. 610-623).





*
“In §4, we discuss how large datasets based on texts from the Internet overrepresent hegemonic viewpoints and encode biases potentially damaging to marginalized populations.”

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 10.1 - Unsupervised Learning

Principal Component Analysis - Variance Maximization

Erik Bekkers

(Bishop 12.1.1)

Recap: see videos

Image credit: Kirillm | Getty Images

Slide credits: Thomas Kipf and Giorgio Patrini

Continuous latent space

- Dimensionality reduction: model the data in a low dim. space

- Example: take one grey-scale image of “3” and make multiple copies by translation and rotation

Figure: Synthetic “3” dataset (Bishop 12.1)

- Pixel space dimension: 100x100 pixels

- Latent space dimension: 3 = 2 (translations) + 1 rotation

- From the 3 latent variables we could generate all 100x100 pixels!

Example continued

- A more realistic dataset of images will have more degrees of freedom in the latent space, such as:
    - Scaling
    - Digits from 0-9
    - Colors
    - Different hand-writing styles
    - Etc.

  … but still much fewer than 100x100!

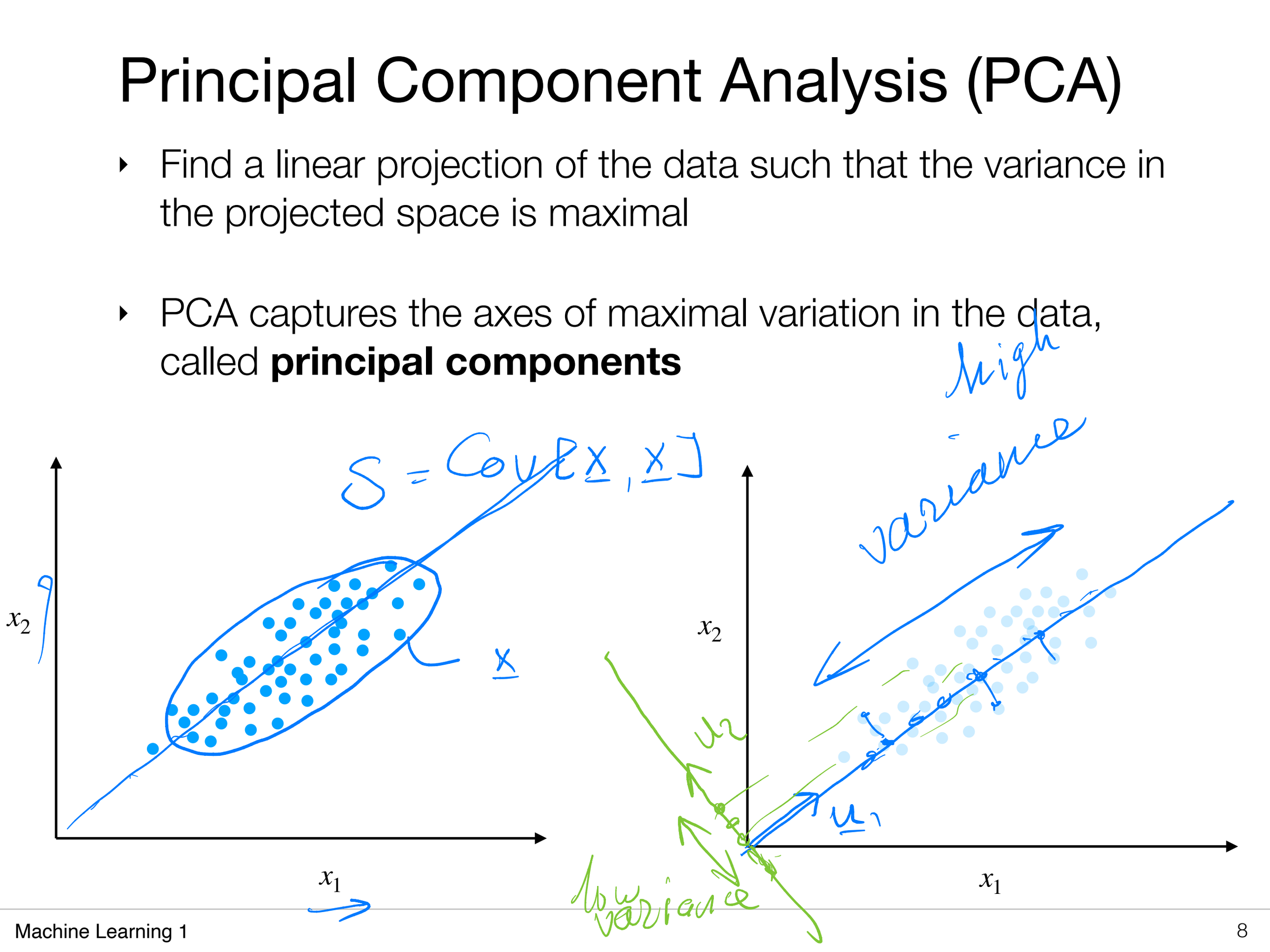
- In this example, the latent subspace represents non-linear transformation of the images

- We first study linear latent spaces with PCA and later consider generalizations to the non-linear case

Principal Component Analysis (PCA)

- Find a linear projection of the data such that the variance in the projected space is maximal

- PCA captures the axes of maximal variation in the data, called **principal components**



Principal Component Analysis (PCA)

- Data: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \;\; \mathbf{x}_n \in \mathbb{R}^D$

- Goal: project data into a $M < D$ dimensional space while **maximizing the variance** of the projected data

- M is given

- Mean and covariance defined by

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T$$

- $\mathbf{S}$ is symmetric and positive semi-definite

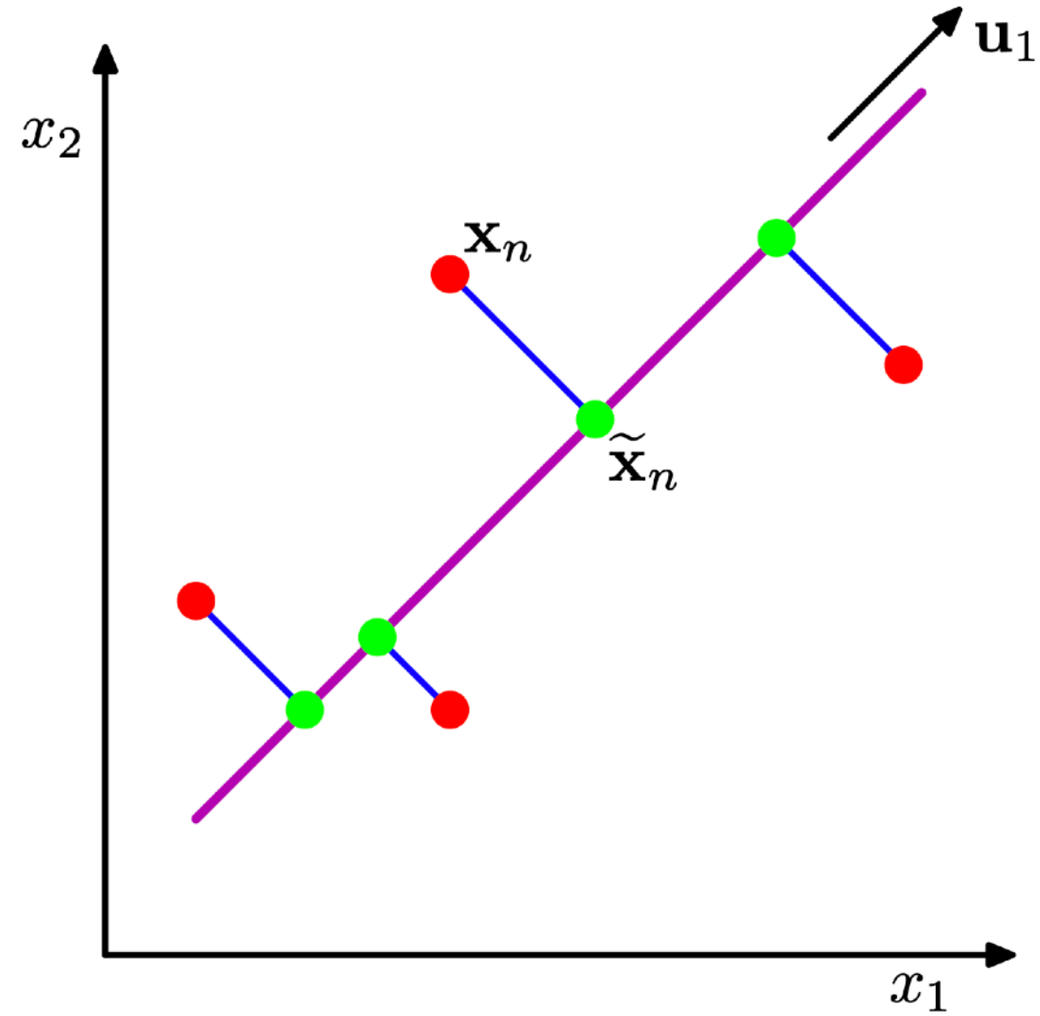


Figure: Maximizing variance of projections (Bishop 12.2)

1D Projection

- Project data int the first latent dimension by a vector $\mathbf{u}_1 \in \mathbb{R}^D$
- The projection gives the scalar $z_1 = \mathbf{u}_1^T \mathbf{x}_n$, the mean of the projection is $\bar{z}_1 = \mathbf{u}_1^T \bar{\mathbf{x}}$
- We only need its direction, so normalize this component: $\|\mathbf{u}_1\|^2 = \mathbf{u}_1^T \mathbf{u}_1 = 1$
- The variance of the projected data is

$$\frac{1}{N}\sum_{n=1}^{N} \underbrace{(\mathbf{u}_1^T \mathbf{x}_n}_{z_{n1}} - \underbrace{\mathbf{u}_1^T \bar{\mathbf{x}})^2}_{z} = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}}))^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} \mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_1$$

$$= \mathbf{u}_1^T \left(\frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \right) \mathbf{u}_1 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

Maximizing the variance of 1 component

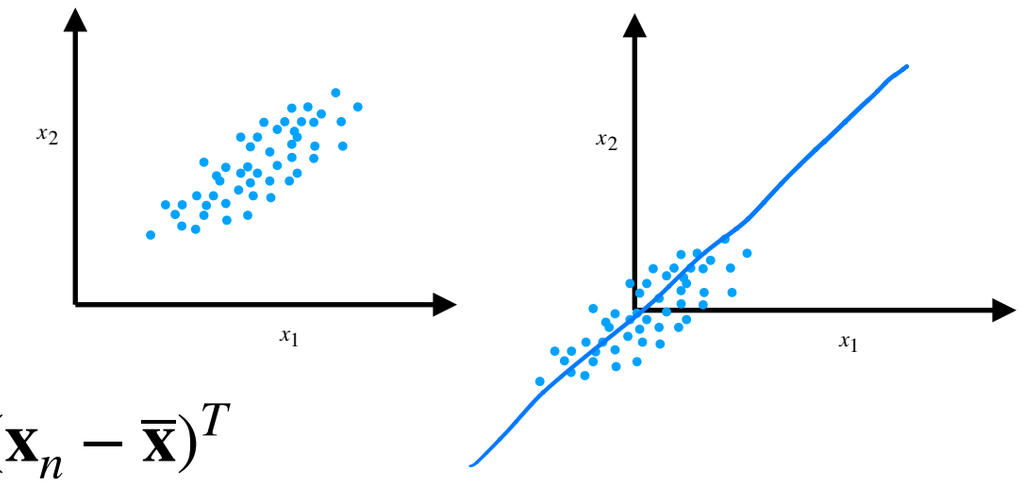
- Solve $\underset{\mathbf{u}_1}{\operatorname{argmax}}\ \mathbf{u}_1^T \mathbf{S}\, \mathbf{u}_1$ subject to $\mathbf{u}_1^T \mathbf{u}_1 = 1$

- Method of Lagrange multipliers
  - Define Lagrangian $L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{S}\, \mathbf{u}_1 + \lambda_1(\mathbf{u}_1^T \mathbf{u}_1 - 1)$
  - Solving for $\mathbf{u}_1$ means $\frac{\partial}{\partial \mathbf{u}_1} L(\mathbf{u}_1, \lambda_1) = \mathbf{S}\, \mathbf{u}_1 - \lambda_1 \mathbf{u}_1 = 0$
    - We need to solve eigensystem $\Rightarrow$ $\mathbf{S}\, \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$
    - So $\mathbf{u}_1$ and λ_1 are respectively an eigenvector and eigenvalue of $\mathbf{S} \in \mathbb{R}^{D \times D}$!

- The $\mathbf{u}_1$ is called a **principal component**.
- The variance of the projected data is $\mathbf{u}_1^T \mathbf{S}\, \mathbf{u}_1 = \lambda_1$
- Maximizing variance means we search for the eigenvecor with **largest eigenvalue**

PCA via maximum variance

- We repeat the procedure for M orthogonal vectors and get a projection defined by $U_M = [\mathbf{u}_1, \ldots, \mathbf{u}_M] \in \mathbf{R}^{D \times M}$



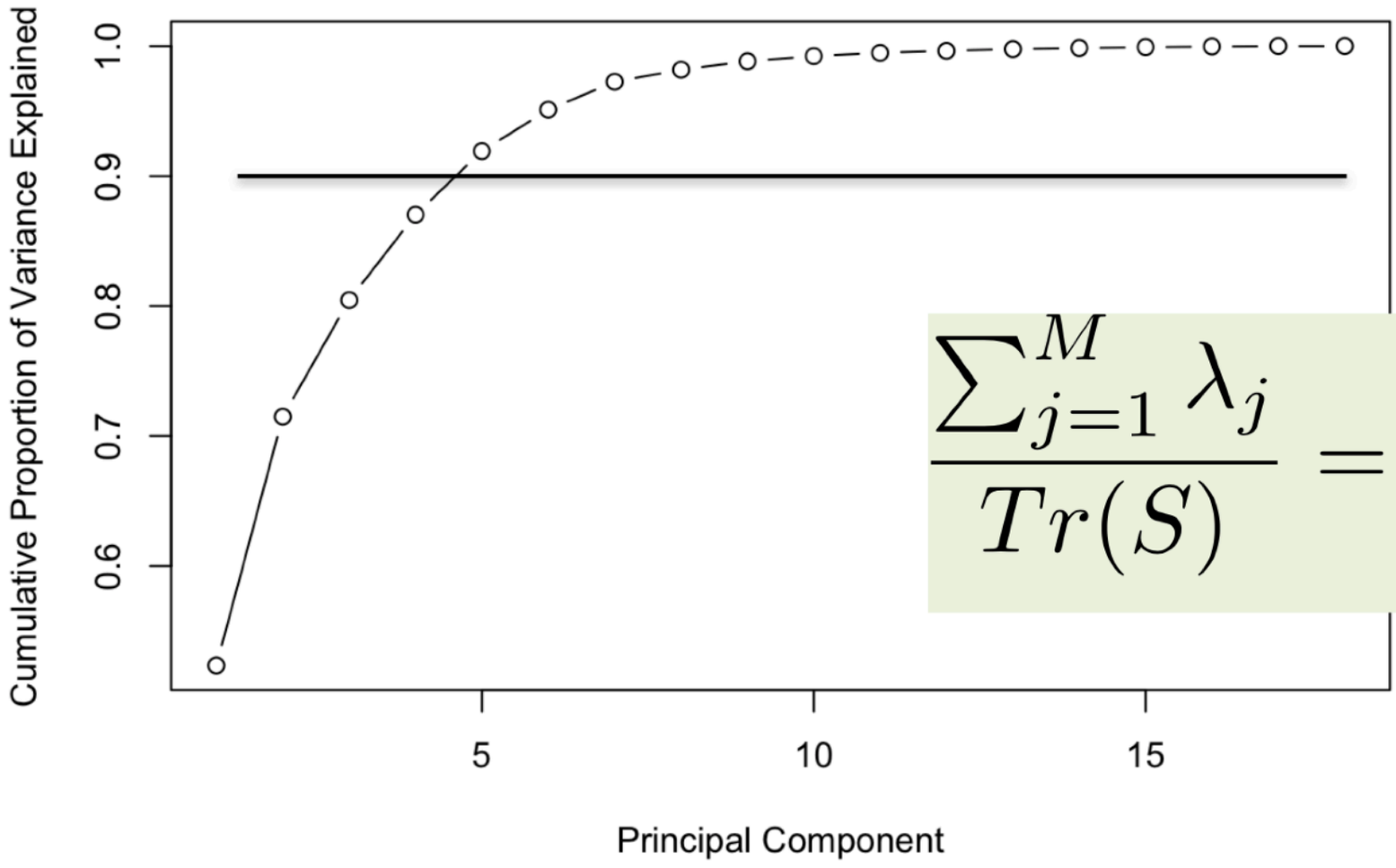
- PCA:
  1. Compute $\overline{\mathbf{x}}$
  2. Compute Covariance $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T$
  3. Compute the M eigenvectors of $\mathbf{S}$ with largest eigenvalue: U_M.
  4. The **projection** to the latent vector then is $\mathbf{z} = \mathbf{U}_M^T(\mathbf{x} - \overline{\mathbf{x}})$ $\approx \begin{pmatrix} z_1 \\ z_2 \\ \vdots \end{pmatrix}$

- U_M contains the M eigenvectors of $\mathbf{S}$, **the principal components**. The eigenvalues are $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_M$
- The matrix $\mathbf{S}$ is positive semi-definite, thus $\forall_j : \lambda_j \geq 0$
- The (total) variance of the projected data is $\mathrm{Tr}[\mathrm{Cov}[\mathbf{z}]] = \sum_{j=1}^{M} \lambda_j$

How to choose M?

- We can measure the discarded variance

- For example to preserve 90% of the variance, pick M such that



The proportion of explained variance

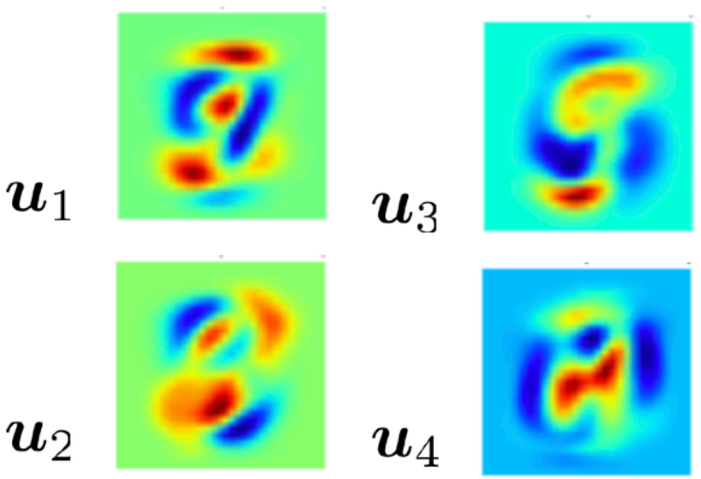
$$\frac{\sum_{j=1}^{M} \lambda_j}{Tr(S)} = \frac{\sum_{j=1}^{M} \lambda_j}{\sum_{j=1}^{D} \lambda_i} > 0.9$$

Applications: dimensionality reduction

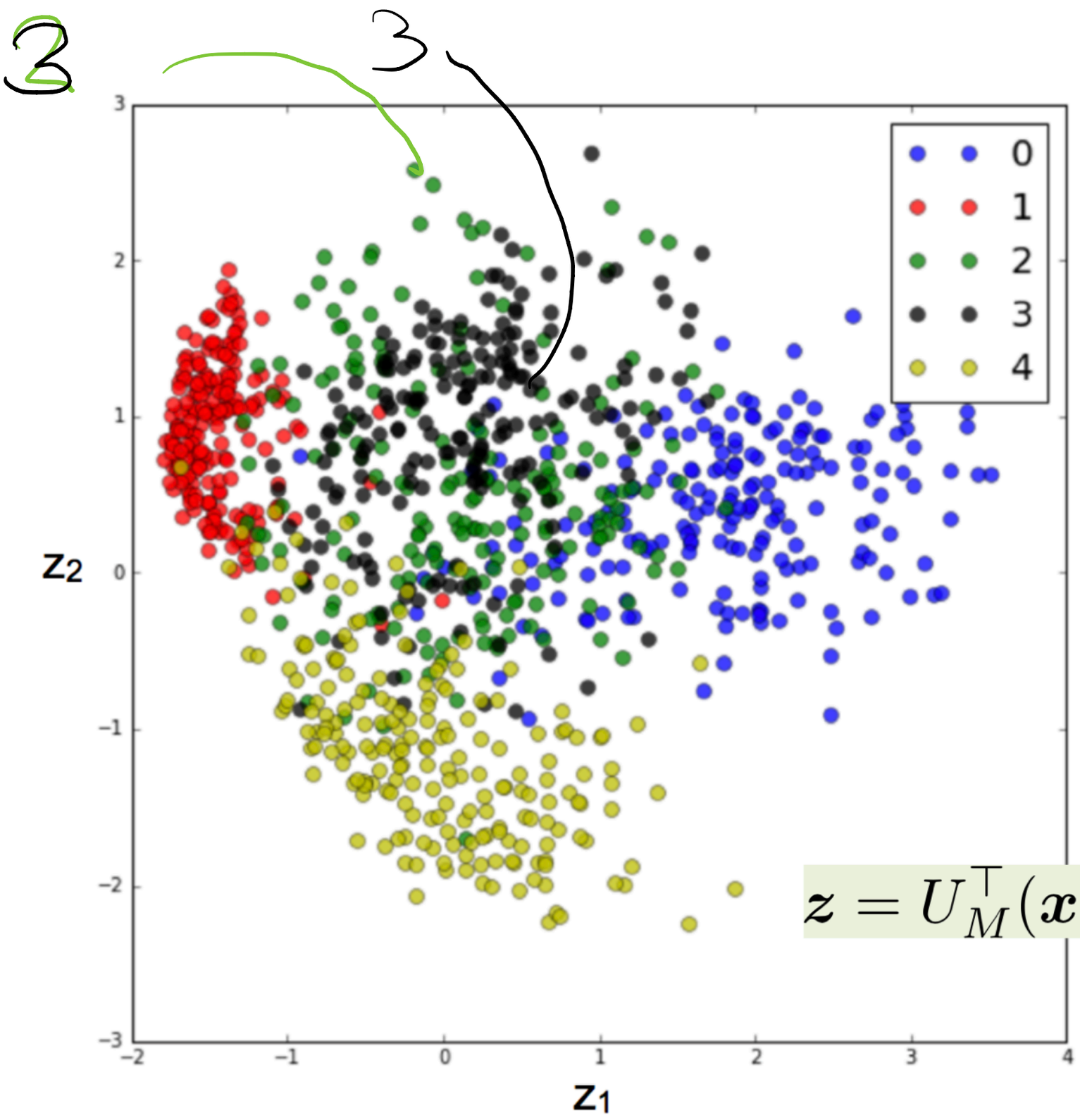
- When data is defined in high dimension (large D) we want to project down to lower dimension because:
  - Reduce time and storage space required
  - For classification/regression: our model **will have less parameters**, thus we need less data points for learning
- Other methods (not covered): **feature selection**. PCA is known as a **feature extraction** method instead.

Applications: 2D Visualization (MNIST)

MNIST: 24 x 24 pixels



Eigenvectors



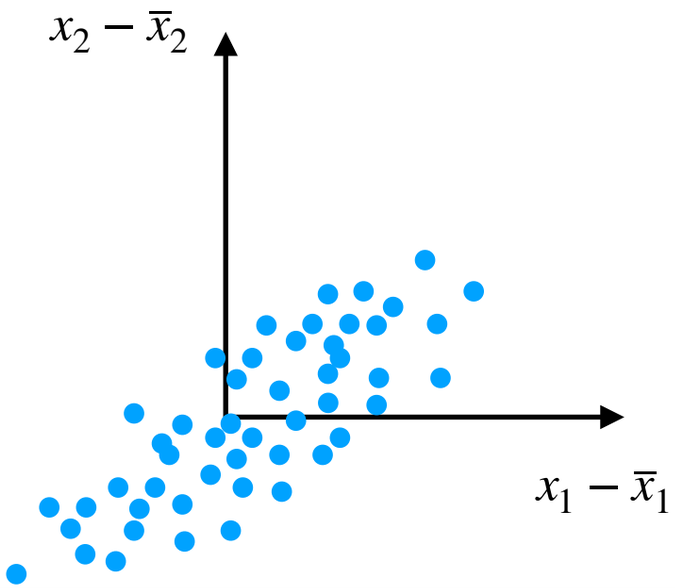
Feature Decorrelation

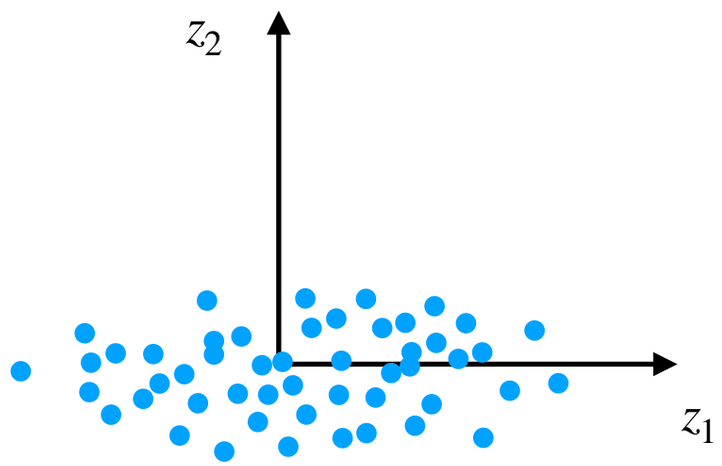
- Good side effect of PCA: features have **no correlation** in the projected space.

- The covariance matrix of the projected data is **diagonal**

$$\frac{1}{N}\sum_{n=1}^{N}\mathbf{z}_n\mathbf{z}_n^T = \frac{1}{N}\sum_{n=1}^{N}\mathbf{U}_M^T(\mathbf{x}_n-\overline{\mathbf{x}})(\mathbf{x}_n-\overline{\mathbf{x}})^T\mathbf{U}_M$$

$$= \mathbf{U}_M^T\mathbf{S}\,\mathbf{U}_M = \mathbf{U}_M^T\mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^T\,\mathbf{U}_M = \Lambda$$





Applications: whitening (or sphering)

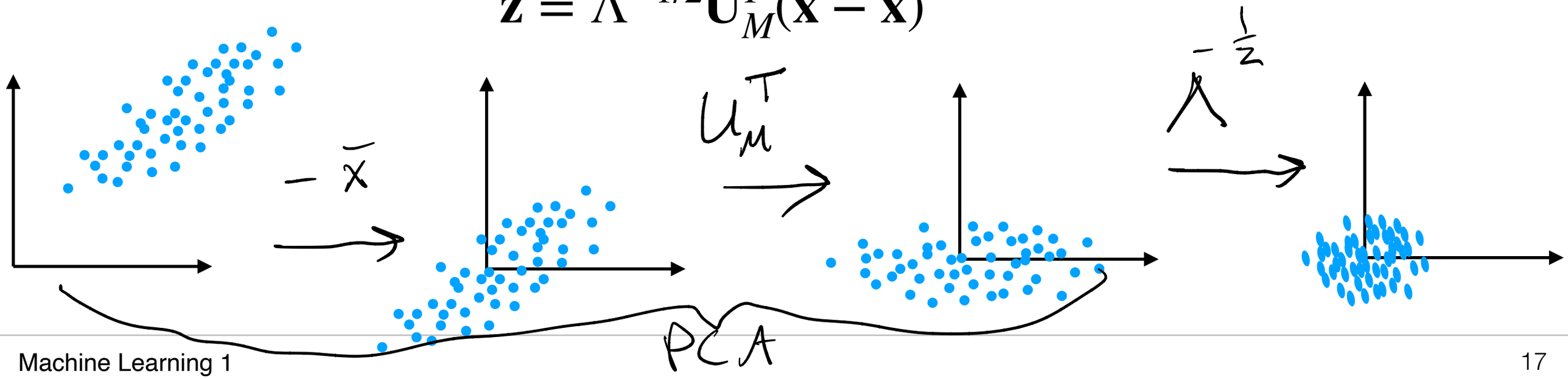
- Before applying learning algorithms we usually do some pre-processing:
  - e.g. **standardization**: subtract the mean and divide by the standard deviation

- With PCA we can **whiten** the data, one step more:
  - **Centre** and **de-correlate** the features:

$$\mathbf{z} = \mathbf{U}_M^T(\mathbf{x} - \bar{\mathbf{x}})$$

  - Cast features to **unit standard deviation** by rescaling:

$$\mathbf{z} = \Lambda^{-1/2}\mathbf{U}_M^T(\mathbf{x} - \bar{\mathbf{x}})$$



University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

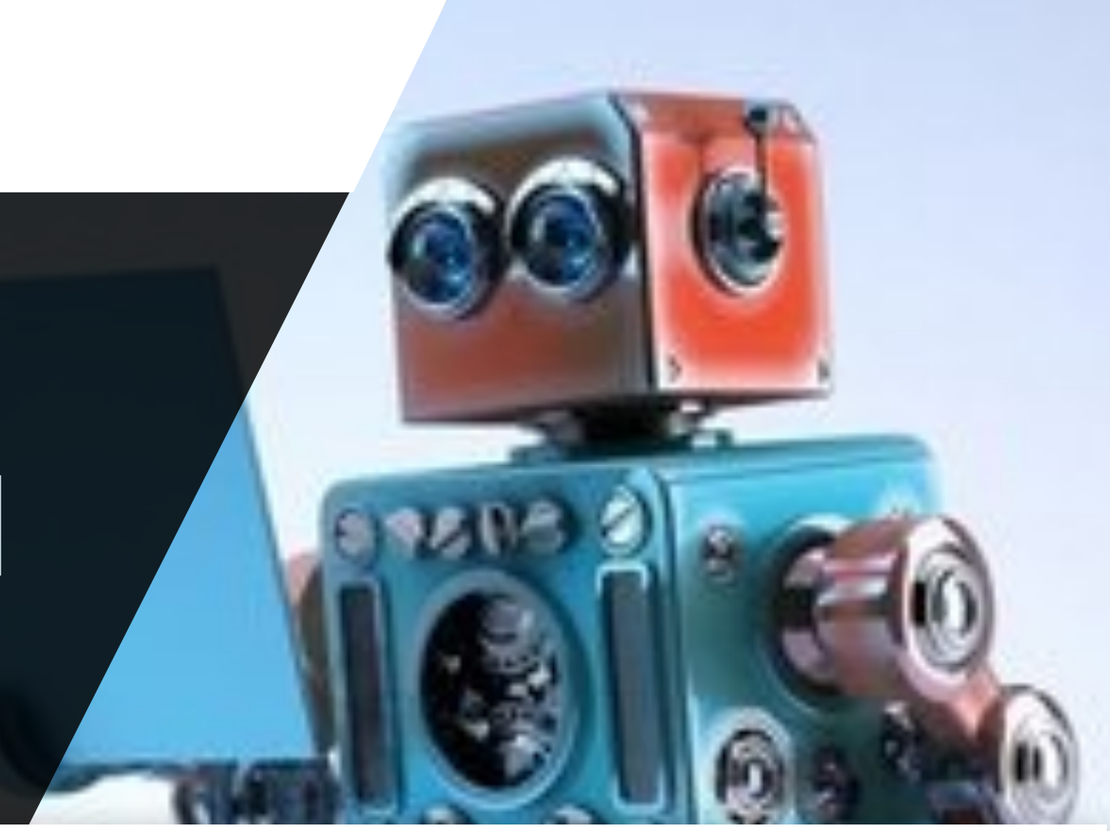
Lecture 10.2 - Unsupervised Learning

Principal Component Analysis - Reconstruction Error Minimization

Erik Bekkers

(Bishop 12.1.2, 12.1.3)

Recap see video's for full slides



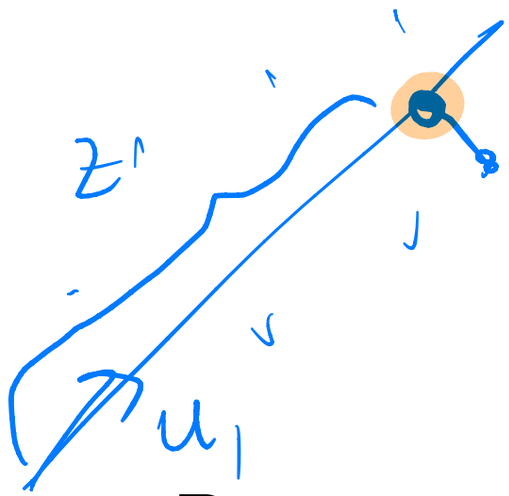
Slide credits: Thomas Kipf and Giorgio Patrini

Image credit: Kirillm | Getty Images

PCA via minimal reconstruction error

- Same method, alternative view to the maximal variance viewpoint

- PCA can be obtain by minimizing the reconstruction error. Find a transformation that minimizes:



$$\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|$$

$$\tilde{x}_n = \bar{x} + u_M z_n$$

- Represent the points in a different orthonormal basis (unknown for now):

$$\{\mathbf{u}_i : \mathbf{u}_i^T\mathbf{u}_i = 1\}_{i=1}^{D} \qquad \mathbf{u}_i^T\mathbf{u}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

PCA via minimal reconstruction error

- We got: $\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \sum_{i=M+1}^{D}\mathbf{u}_i^T\mathbf{S}\,\mathbf{u}_i = \sum_{i=M+1}^{D}\lambda_i$
- Solve for $\mathbf{u}_i$ under constraint $\mathbf{u}_i^T\mathbf{u}_i = 1$ (else we get $\mathbf{u}_i = \mathbf{0}$)
- Method of Langrange multipliers → Solve eigensystem
- Find M eigenvectors with largest eigenvalues, such that the remaining $(D - M)$ are the smallest

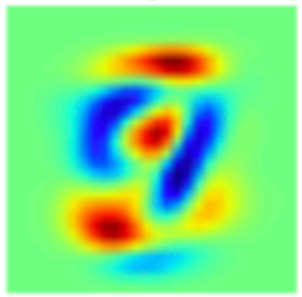
min reconstruction error = max projected variance

Applications: compression (MNIST)

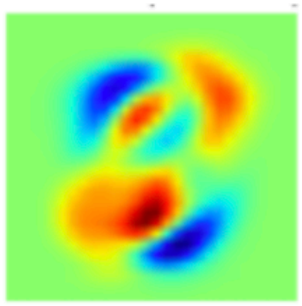
Eigenvectors:

Reconstructions: $\tilde{\mathbf{x}} = \mathbf{U}_M \mathbf{z} + \overline{\mathbf{x}}$

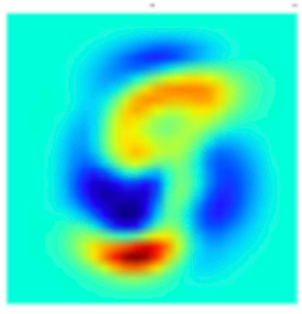
$\boldsymbol{u}_1$



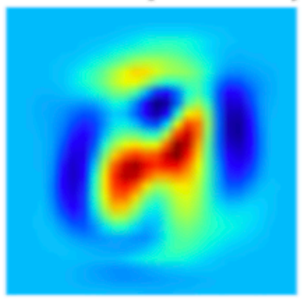
$\boldsymbol{u}_2$

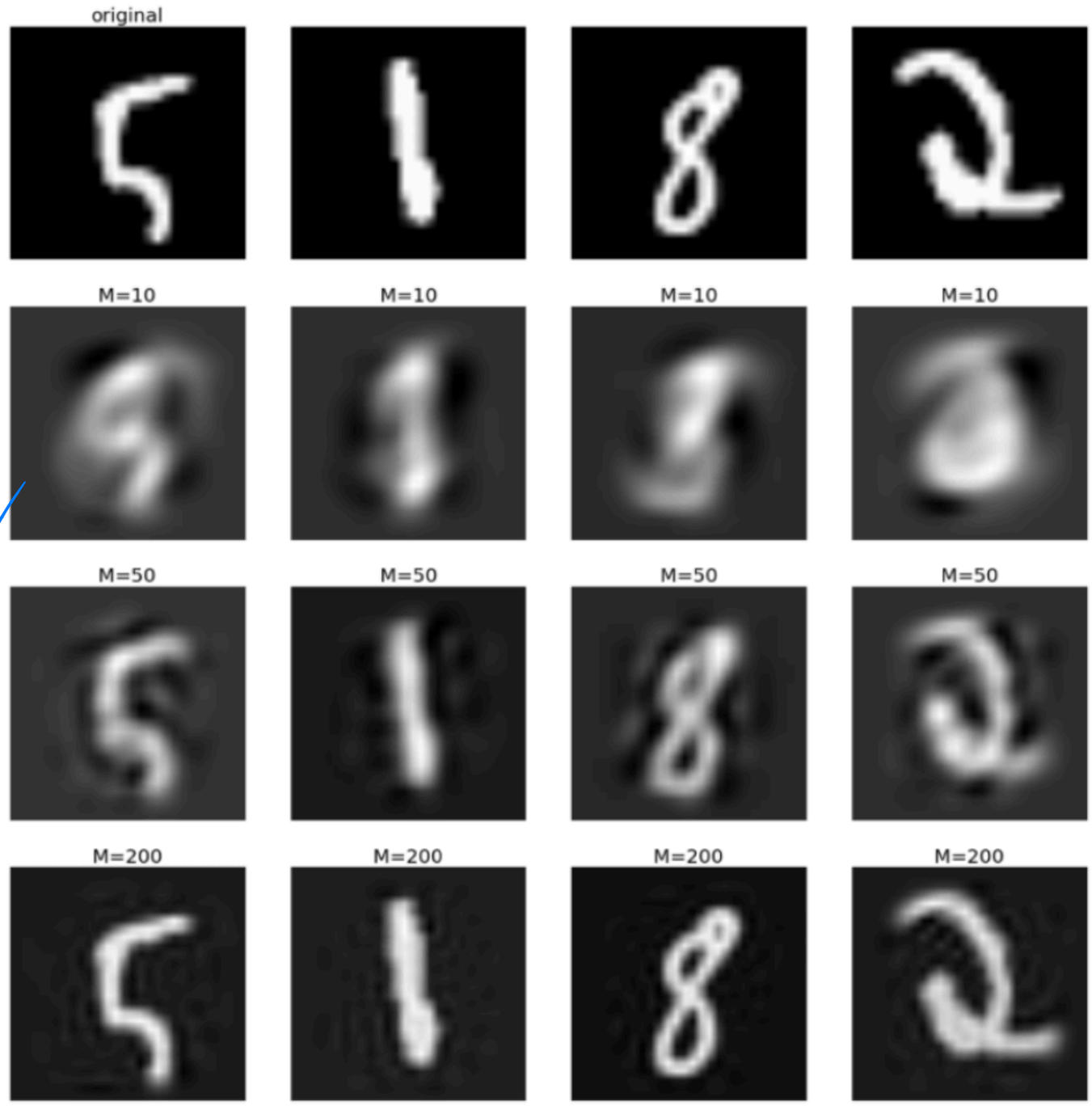


$\boldsymbol{u}_3$



$\boldsymbol{u}_4$





University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 10.4 - Unsupervised Learning

Non-linear PCA

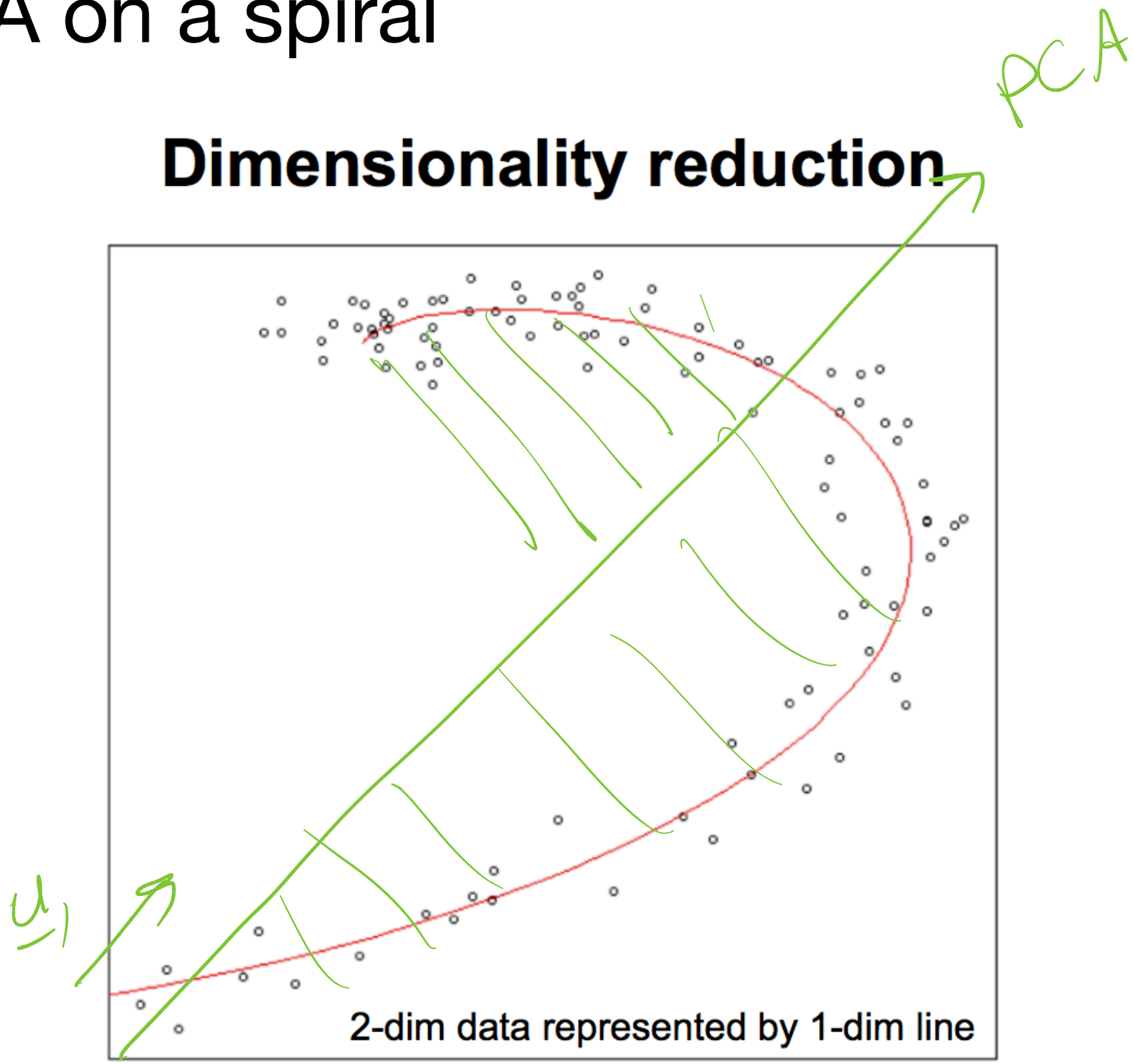
Erik Bekkers

(Bishop 12.3, (12.4.1), 12.4.2)

Image credit: Kirillm | Getty Images

Slide credits: Thomas Kipf and Giorgio Patrini

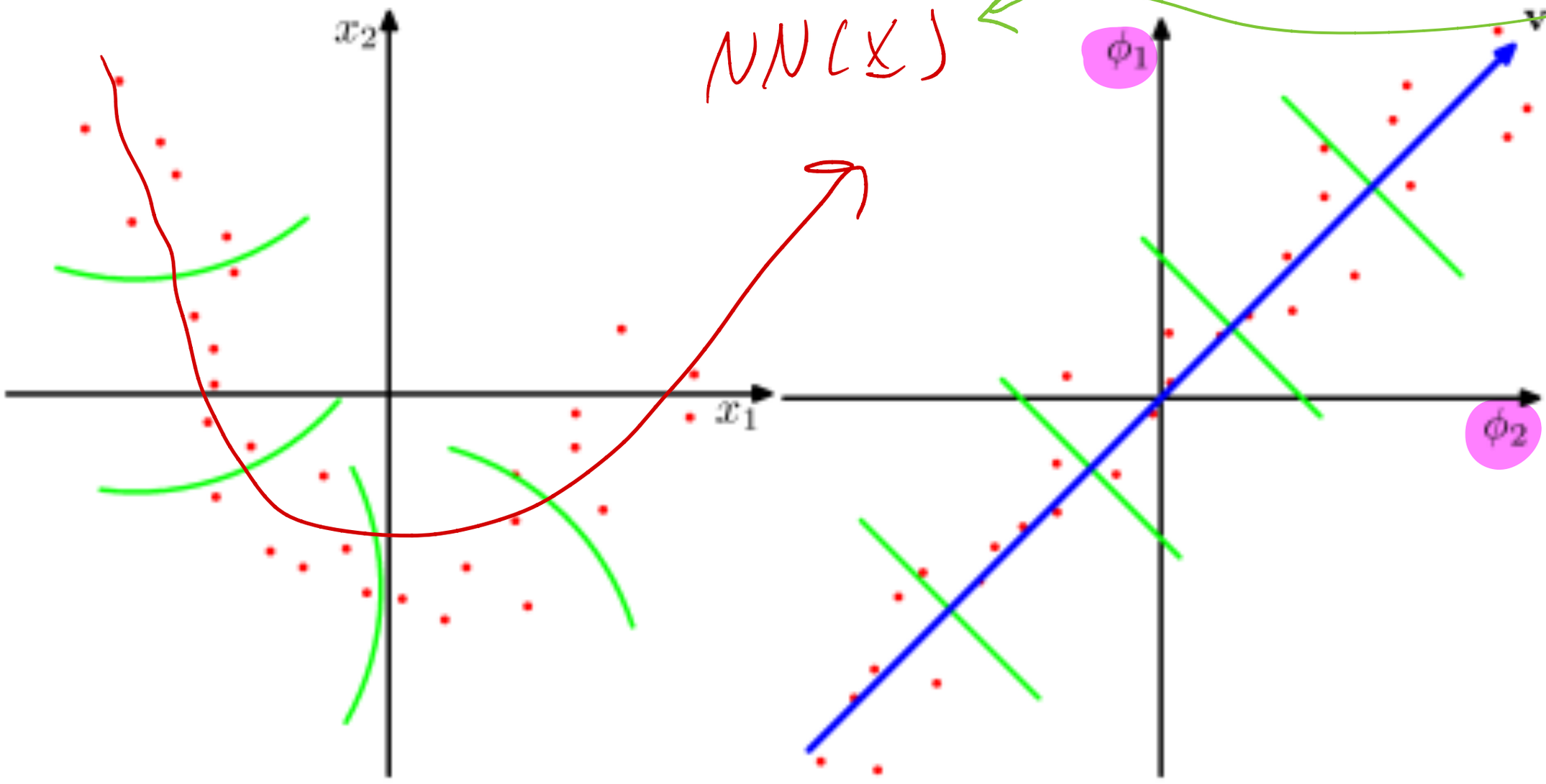
PCA on a spiral



PCA using basis functions

but how do we pick the basis funcs?

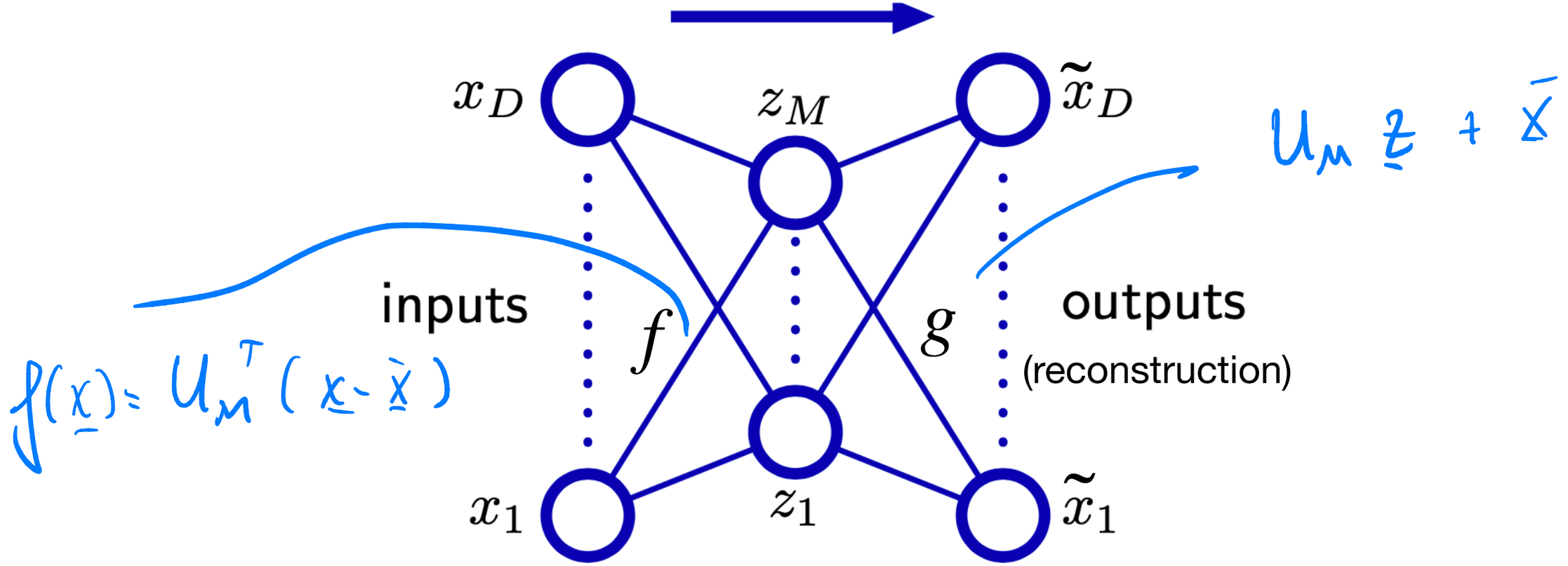
NN(x)



(Bishop 12.16)

Auto-encoders (auto-associative neural nets)

- Non-linear dimensionality reduction with **neural networks**
- Maps:
  - Encoder (projection): $\mathbf{z} = f(\mathbf{x})$
  - Decoder (reconstruction): $\tilde{\mathbf{x}} = g(\mathbf{z})$
  - Both are neural networks
- Goal: minimize the reconstruction error



(Bishop 12.18)

Autoencoder objective

- Minimize the error between original and reconstructed input:

not linear

$$\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{x}_n - g_{\mathbf{w}'}(f_{\mathbf{w}}(\mathbf{x}_n))\|^2$$

- Non-linear, no closed form solution, solve via SGD!

- Recall minimum error viewpoint of PCA:

$$f(\mathbf{x}) = \mathbf{U}_{\mathbf{M}}^T(\mathbf{x} - \bar{\mathbf{x}}), \qquad g(\mathbf{x}) = U_{\mathbf{M}}\mathbf{z} + \bar{\mathbf{x}}$$

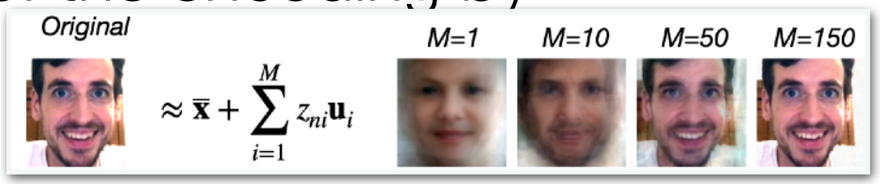
"Wx + b"

- PCA <—> 2 layer autoencoder without activation functions

$W = U_M^T \quad b = U_M^T \bar{x}$

Autoencoder as generator

- As before, we could judge visually the quality of the encoding by looking at reconstructed images



- Bonus: we can use $\tilde{\mathbf{x}} = g(\mathbf{z})$ as a **generator of fake data** (images):
  - Train the autoencoder to get $\tilde{\mathbf{x}} = g(\mathbf{z})$
  - Sample* a random $\mathbf{z}$ $\sim p(z)$
  - Feed it into the generator to get fake image $\tilde{x} = g(\mathbf{z})$

Recall PPCA

$p(z|x)$ now untractable so approx with encoder $q(z|x) = NN(x) = f(x)$

A probabilistic model needs to be defined before sampling, this definition needs to be part of the encoding/decoding optimization pipeline, see* ***Variational Autoencoders* *(VAE) [Kingma & Welling, ICLR 2014]*

Variational Autoencoder on MNIST

$z \sim p(z)$

Which one is real? $\hat{x} = g(z)$

… autoencoders in 2018

Trained on CelebA in [Patrini et al., Sinkhorn AutoEncoders, 2018]

Kernel PCA

Bishop 12.3

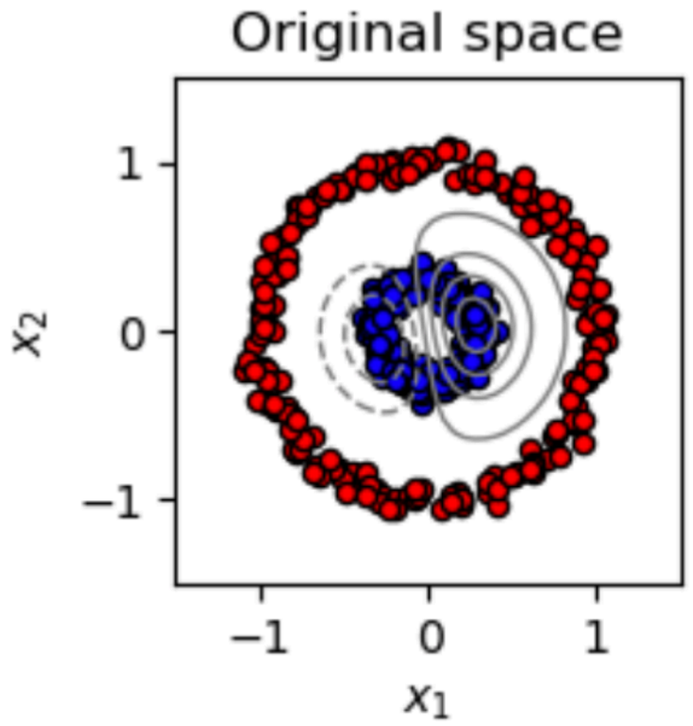
- Use feature transformations to “linearize” the data
  - Via some specific choice of basis functions $\boldsymbol{\phi}(\mathbf{x}_n)$
  - Do PCA in this space

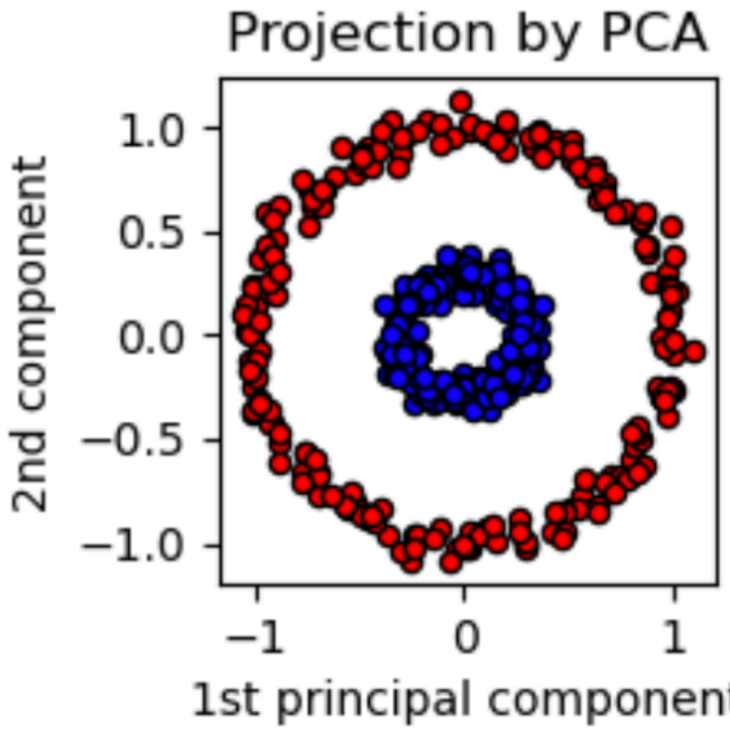
$$S = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T, \qquad C = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T$$

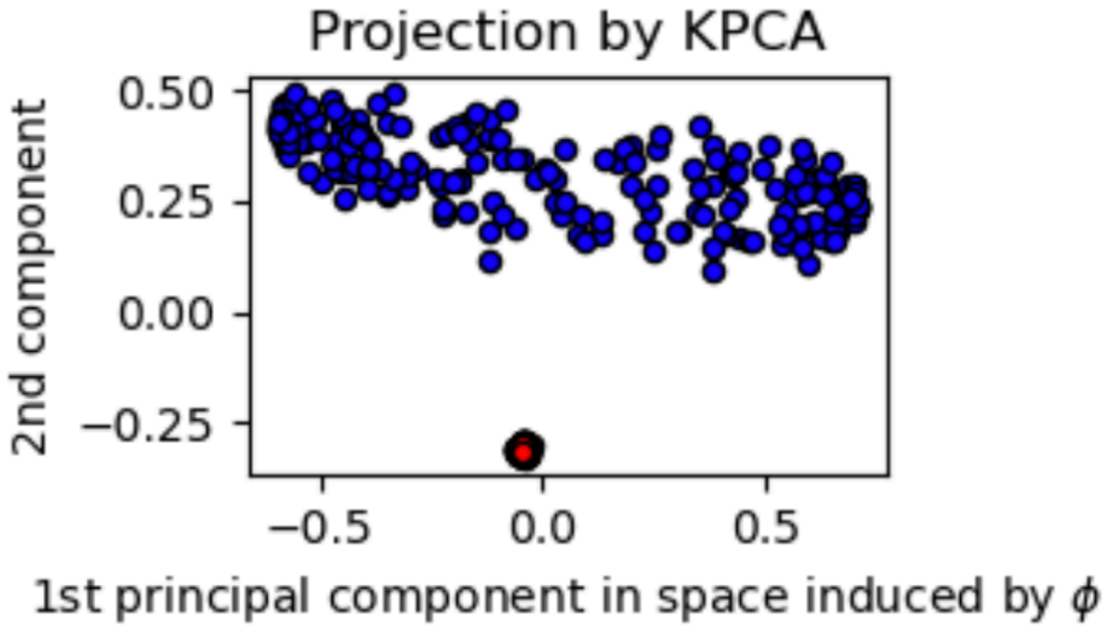
- Kernel approach:
  - Let $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\phi}(\mathbf{x}_j)$ be the kernel associated with the basis functions
  - Let $\mathbf{u}_i$ be the principal components of C
  - Let $z_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^T\mathbf{u}_i$ the projection onto the i^{th} component
  - Let $\mathbf{a}_i$ be the i^{th} eigenvector of $\mathbf{K} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$
  - Then $z_i(\mathbf{x}) = \sum_{n=1}^{N} a_{in} k(\mathbf{x}, \mathbf{x}_n)$ - The projection is purely in terms of the other data points via k!
- Kernel trick: use some defined kernel $k(\mathbf{x}_i, \mathbf{x_j})$ without explicitly defining the basis functions $\boldsymbol{\phi}$

we've seen this same but in terms of k

Example







- With RBF
- $k(\mathbf{x_i}, \mathbf{x_j}) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}$
- Source:
  Kernel-PCA from
  http://scikit-learn.org

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 11 - Kernel Methods

The Kernel Trick - Support Vector Machines - Inequality Constraint Optimization - Kernel SVM

Erik Bekkers

Catch-up:
Probabilistic PCA
Non-linear PCA

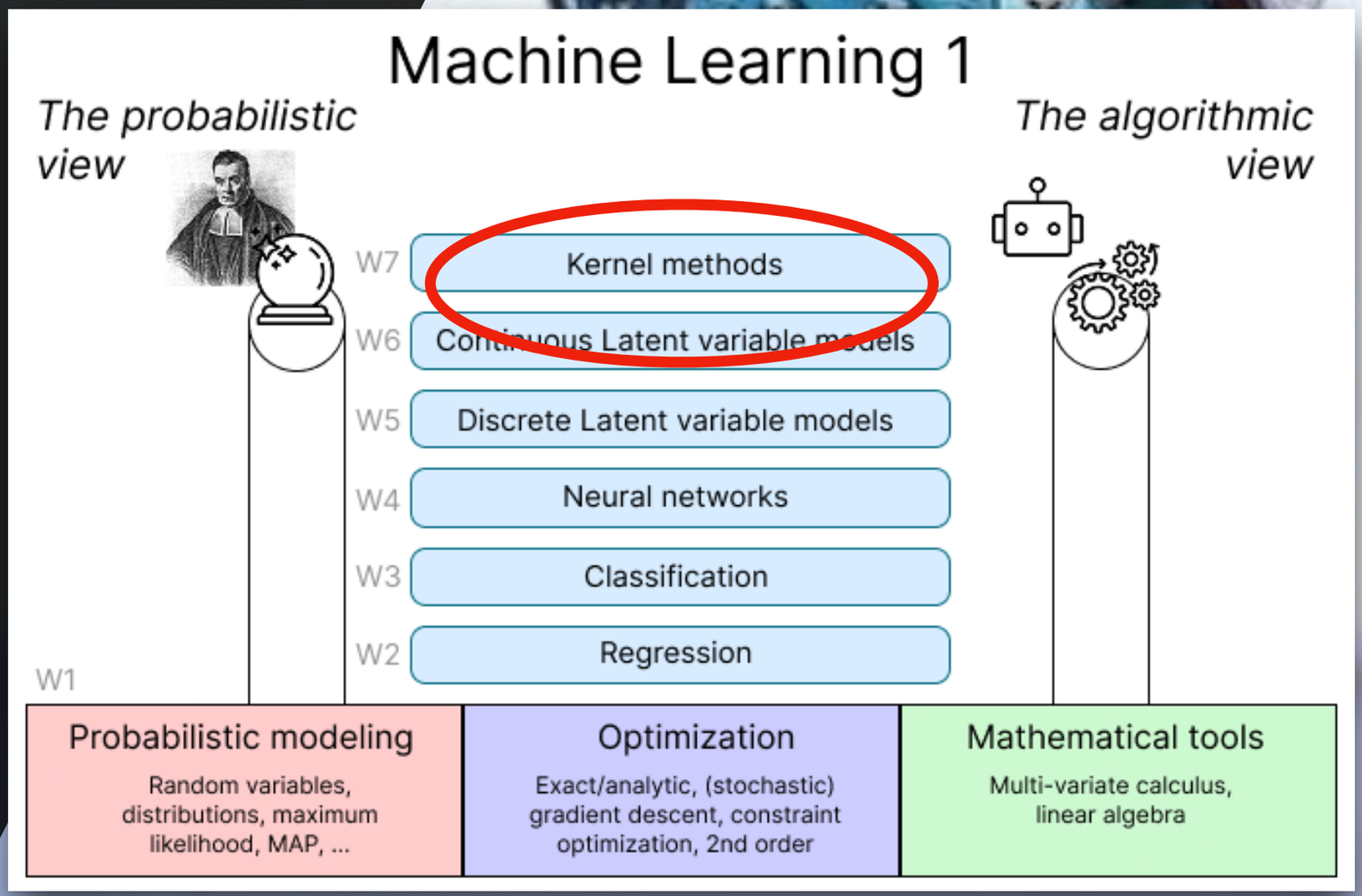


Image credit: Kirillm | Getty Images

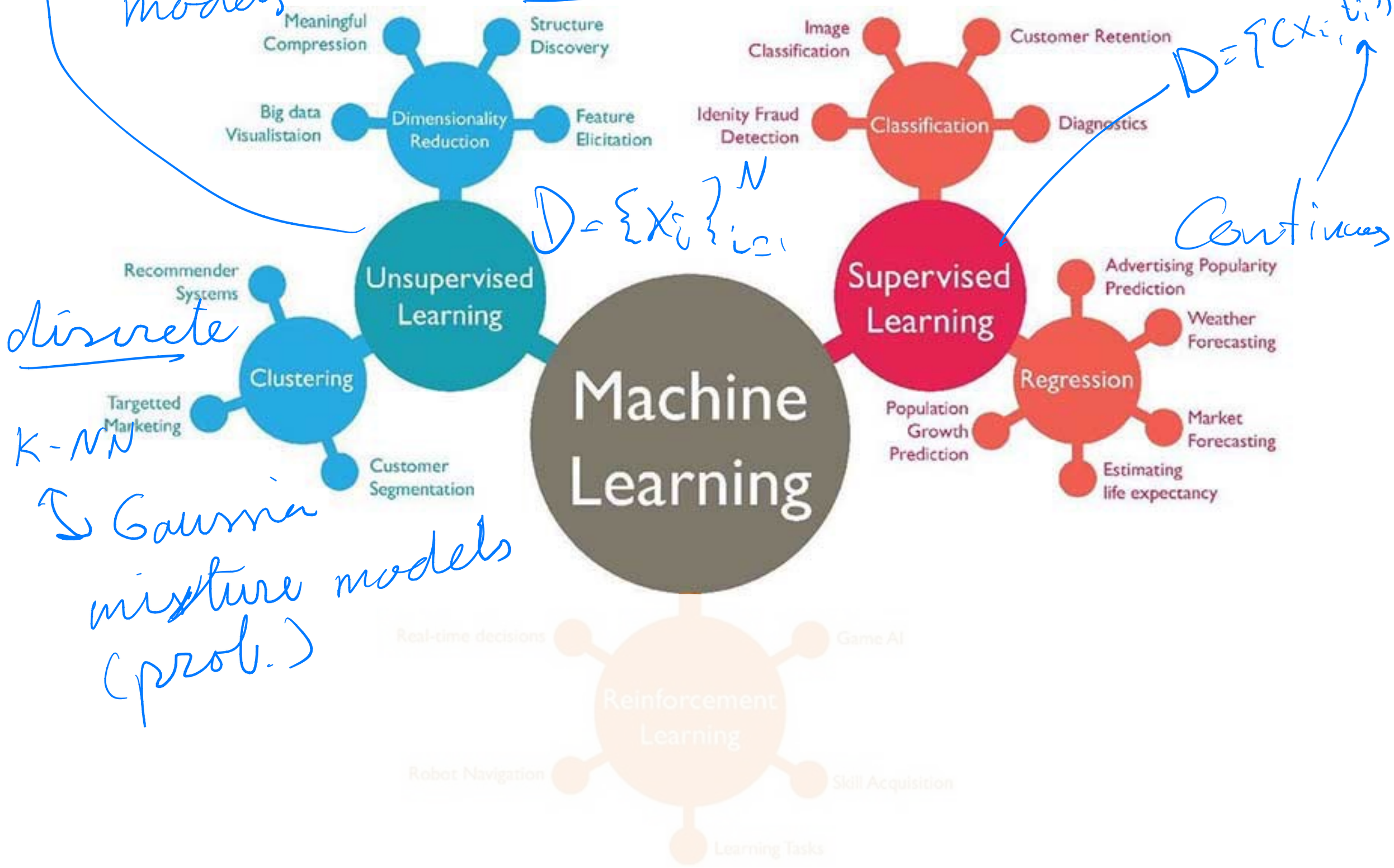
Latent variable models

continuous

Discrete

$D = \{x_i\}_{i=1}^{N}$

Continuous



discrete

K-NN

↳ Gaussian mixture models (prob.)

Image source : www.techleer.com

Tasks:

- Regression
- Classification
- Latent-variable modeling

$p(z \mid x, w)$

$p(x \mid z)$

Probabilistic modeling:

- Define (predictive) distributions
- Find model parameters via
- ML, MAP (or use fully Bayesian)

Discriminative methods:

- more algorithmic in nature
- no probabilities…

$p(t \mid x, \underline{w})$

prior $p(\underline{w}) \rightarrow$ posterior $p(\underline{w} \mid X, \underline{t})$

Image source : www.techleer.com

Tasks:
- Regression
- Classification
- Latent-variable modeling

$$y(x) = \sum_n a_n k(x, x_n)$$

Probabilistic modeling:
- Define (predictive) distributions
- Find model parameters via
- ML, MAP (or use fully Bayesian)

Discriminative methods:
- more algorithmic in nature
- no probabilities…

Non-parametric models:
- Kernel methods
- Gaussian processes

Parametric model class:
- Linear models $\mathbf{y}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- Generalized $\mathbf{y}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
- Neural Networks
- Distribution classes
  (Gaussian, Bernoulli, …)
- Basis functions

Image source : www.techleer.com

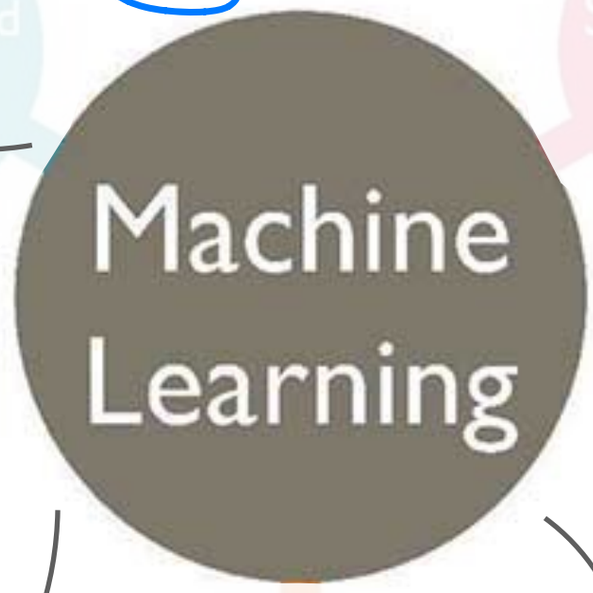
Optimization:

- Convex vs non-convex optimization
- Find stationary points (solve $\frac{\partial E}{\partial w} = 0$)
- Analytic solutions vs numerical (e.g. SGD)
- Method of Lagrange multipliers:
  - Equality constraint optimization
  - Inequality constraint optimization

Tasks:

- Regression
- Classification
- Latent-variable modeling

machine learning is optimization



Probabilistic modeling:

- Define (predictive) distributions
- Find model parameters via
- ML, MAP (or use fully Bayesian)

Discriminative methods:

- more algorithmic in nature
- no probabilities…

Non-parametric models:

- Kernel methods
- Gaussian processes

Parametric model class:

- Linear models $\mathbf{y}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- Generalized $\mathbf{y}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
- Neural Networks
- Distribution classes
  (Gaussian, Bernoulli, …)
- Basis functions

Image source : www.techleer.com

University of Amsterdam
Informatics Institute

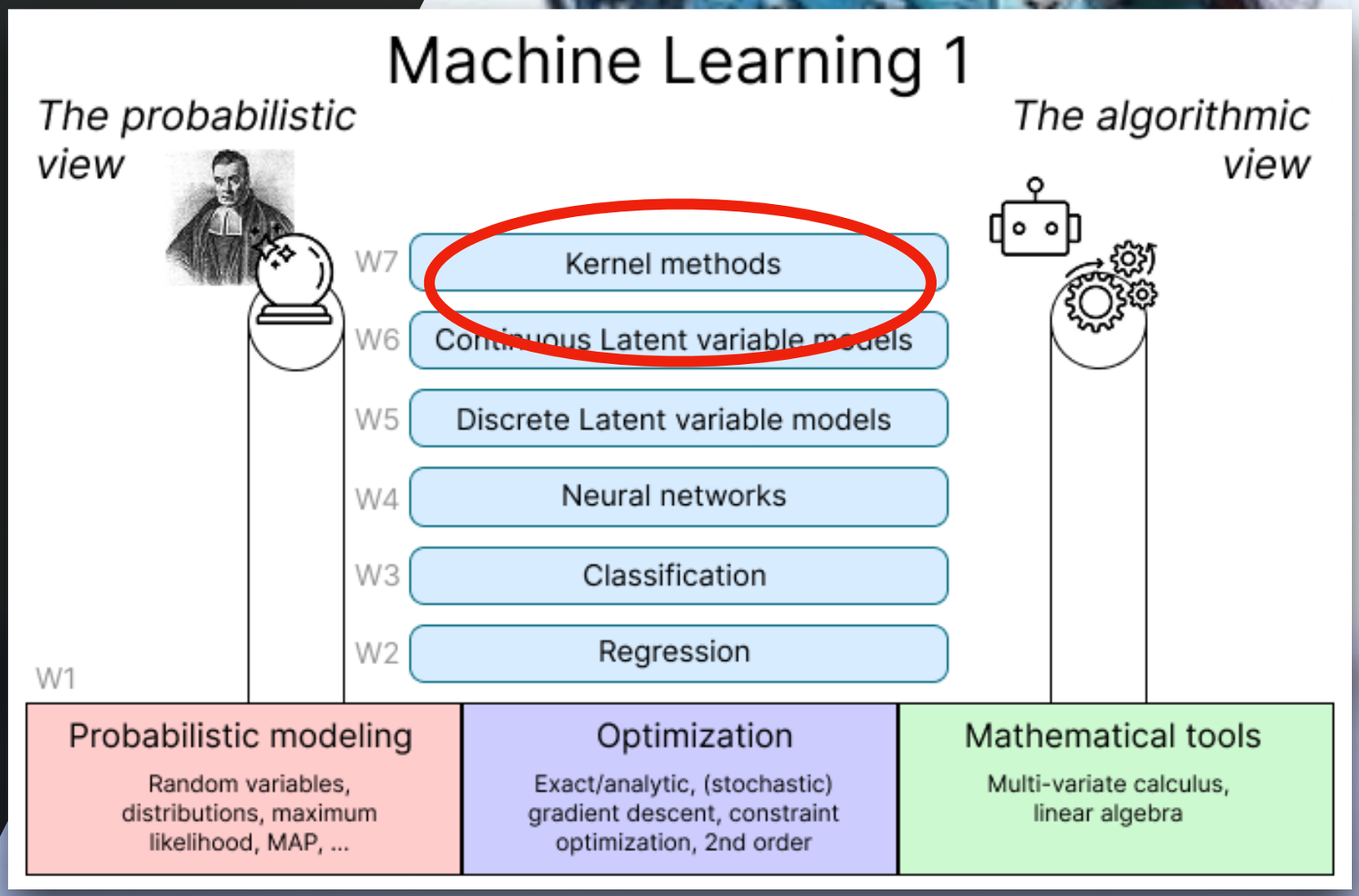
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 11.1 - Kernel Methods
Kernelizing Linear Models

Erik Bekkers

(Bishop 6.0, 6.1)



Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

So Far: Parametric Models

- Fixed basis function methods: $\boldsymbol{\phi}(\boldsymbol{x}) = \left(\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_{M-1}(\boldsymbol{x})\right)^T$
  - Linear regression: $y = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b$
  - Linear models for classification: $y = f(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b)$

- Learnable basis functions: Neural networks

$$y(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{w}^{(2)}) = \sum_{m=0}^{M} w_m^{(2)} h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)$$

Optimize → $\mathbf{w}^*$

- Training:
  - MLE, MAP: use training data to obtain point estimate of $\mathbf{w}$
  - Full Bayesian: use training data to obtain posterior $p(\mathbf{w} \mid \mathbf{X}, \mathbf{t})$

$\mathbf{w}_D^*$ is in a sense the data is encoded in solution $\mathbf{w}_D^*$

- Test time: Discard training data, only need $\mathbf{w}$ or $p(\mathbf{w} \mid \mathbf{X}, \mathbf{t})$

Parametric vs Non-Parametric Models

- Parametric models = models with a finite number of parameters
- Non-Parametric models = models with no explicitly defined parameters (but *implicitly still* work with (finite) or infinite number of parameters)
- Parametric methods:
  - Working in the (finite dimensional) parameter space
- Non-Parametric methods
  - Directly working in possibly infinite dimensional function spaces
  - Typically we have $M \gg N$

Non-Parametric Kernel Methods

- Kernel methods: Use (subset of) training points for predictions (test time!). Useful if $M \gg N$

- Linear parametric models:
  - Can be re-cast into equivalent ‘dual representation’
  - Predictions are based on linear combinations of the *kernel function* evaluated at training data points

Today’s example

Linear model obtained with Ridge regression

Primal case: $y(\mathbf{x}', \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}')$ with primal variable $\mathbf{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda \mathbf{I}_M\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$

Dual case: $y(\mathbf{x}', \mathbf{a}) = \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}')$ with dual variables $\boldsymbol{a} = \left(\boldsymbol{K} + \lambda \mathbf{I}_N\right)^{-1} \boldsymbol{t}$

$D_{train} = \{x_n\}_{n=1}^{N}$

Non-Parametric Kernel Methods

- Kernel methods: Use (subset of) training points for predictions (test time!). Useful if $M \gg N$

- Linear parametric models:
  - Can be re-cast into equivalent ‘dual representation’
  - Predictions are based on linear combinations of the *kernel function* evaluated at training data points

- For linear models with fixed feature vectors $\boldsymbol{\phi}(\mathbf{x})$ we will encounter

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$$

- Kernel measures similarity between $\mathbf{x}$ and $\mathbf{x}'$ in feature space defined by mapping $\boldsymbol{\phi}(\mathbf{x})$ and the kernel is symmetric

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$$

Kernelized Ridge Regression

- Goal: Minimize sum of squared errors with quadratic weight penalty

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Solution: Solve $\dfrac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 0$ for $\mathbf{w}$:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \sum_{n=1}^{N}\{\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n) - t_n\}\boldsymbol{\phi}(\mathbf{x}_n)^T + \lambda\mathbf{w}^T = \mathbf{0}$$

$$\Leftrightarrow \quad \mathbf{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T + \lambda\mathbf{I}\right) = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^T$$

$$\Leftrightarrow \quad \left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T + \lambda\mathbf{I}\right)\mathbf{w} = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)$$

$$\rightarrow \quad \mathbf{w} = \left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T + \lambda\mathbf{I}\right)^{-1}\sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n) = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

Kernelized Ridge Regression

- Goal: Minimize sum of squared errors with quadratic weight penalty

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Solution 1 (via inversion of $M \times M$ matrix):

$$\mathbf{w} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}_M\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{t},$$

$M \times M$

$$\boldsymbol{\Phi}^T\boldsymbol{\Phi} \in \mathbb{R}^{M\times M}$$

- Objective is to try to write in terms of the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$

  Use matrix inversion lemma (see e.g. Bishop C.5):

$$\left(P^{-1} + B^TR^{-1}B\right)^{-1}B^TR^{-1} = PB^T\left(BPB^T + R\right)^{-1}$$

$$\begin{pmatrix} P^{-1} = \lambda\mathbf{I}_M \\ B = \boldsymbol{\Phi} \\ R = \mathbf{I}_N \end{pmatrix}$$

- Solution 2 (via inversion of $N \times N$ matrix)

$N \times N$

$$\mathbf{w} = \boldsymbol{\Phi}^T\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I}_N\right)^{-1}\boldsymbol{t} = \boldsymbol{\Phi}^T\left(\boldsymbol{K} + \lambda\mathbf{I}_N\right)^{-1}\boldsymbol{t}$$

with Gramm matrix $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ with $K_{ij} = \boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$

Kernelized Ridge Regression

- Goal: Minimize sum of squared errors with quadratic weight penalty

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- Solution 2 (in kernel form):

$$\mathbf{w} = \boldsymbol{\Phi}^T\left(\boldsymbol{K} + \lambda\mathbf{I}_N\right)^{-1}\boldsymbol{t}$$

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T,\ K_{ij} = \boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$$

dual a

- Primal/dual viewpoint
  - Primal variable: $\mathbf{w} = \boldsymbol{\Phi}^T\mathbf{a}$
  - Dual variable: $\mathbf{a} = \left(\boldsymbol{K} + \lambda\mathbf{I}_N\right)^{-1}\boldsymbol{t}$

$$\arg\min_{\mathbf{w},\mathbf{z}} \frac{1}{2}\|\mathbf{z}\|^2 \text{ with constraints } \mathbf{z} = \boldsymbol{\Phi}\mathbf{w} - \mathbf{t} \text{ and } \frac{1}{2}\|\mathbf{w}\|^2 \le R^2$$

$$\arg\min_{\mathbf{a}} \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{t} + \frac{\lambda}{2}\|\mathbf{a}\|^2$$

- Predictive mean of primal viewpoint $y(\mathbf{x}', \mathbf{w}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}')$ $= \phi^T w = \phi^T \Phi^T a$

  of dual viewpoint $y(\mathbf{x}', \mathbf{a}) = \sum_{n=1}^{N} a_n k(\mathbf{x}_n, \mathbf{x}')$

$\sum a_n \phi^T(x_n)\phi(x)$

$k(x_n, x)$

Primal vs Dual/Kernel Approach

- Computational cost (closed form solutions):
  - The dual variables. $\boldsymbol{a} = \left(\boldsymbol{K} + \lambda \mathbf{I}_N\right)^{-1} \boldsymbol{t}$ $O(N^3)$
  - The primal variables $\mathbf{w} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \mathbf{I}_M\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$ $O(M^3)$
- Computational cost (predictions):
  - Dual case: $y(\mathbf{x}', \mathbf{a}) = \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}')$ $O(NM)$
  - Primal case: $y(\mathbf{x}', \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}')$ $O(M)$
- But… dual approach:
  - No explicit parameters (implicitly many!) -> nonparametric model
  - Does not rely on explicit features but on similarity kernel function.
  - Can be slow at prediction
  - Upcoming: **Kernel methods with sparse solutions**!

Parametric: make predictions with *explicit* weights and basis functions

$$y(x', \mathbf{w}) = \boldsymbol{\phi}(x')^T \mathbf{w}$$

Non-parametric: make predictions using data points and a kernel

$$y(x', \mathbf{a}) = \sum_{n=1}^{D} a_n k(x_n, x')$$

with $k(x_n, x')$ *implicitly* defined by basis functions

University of Amsterdam
Informatics Institute

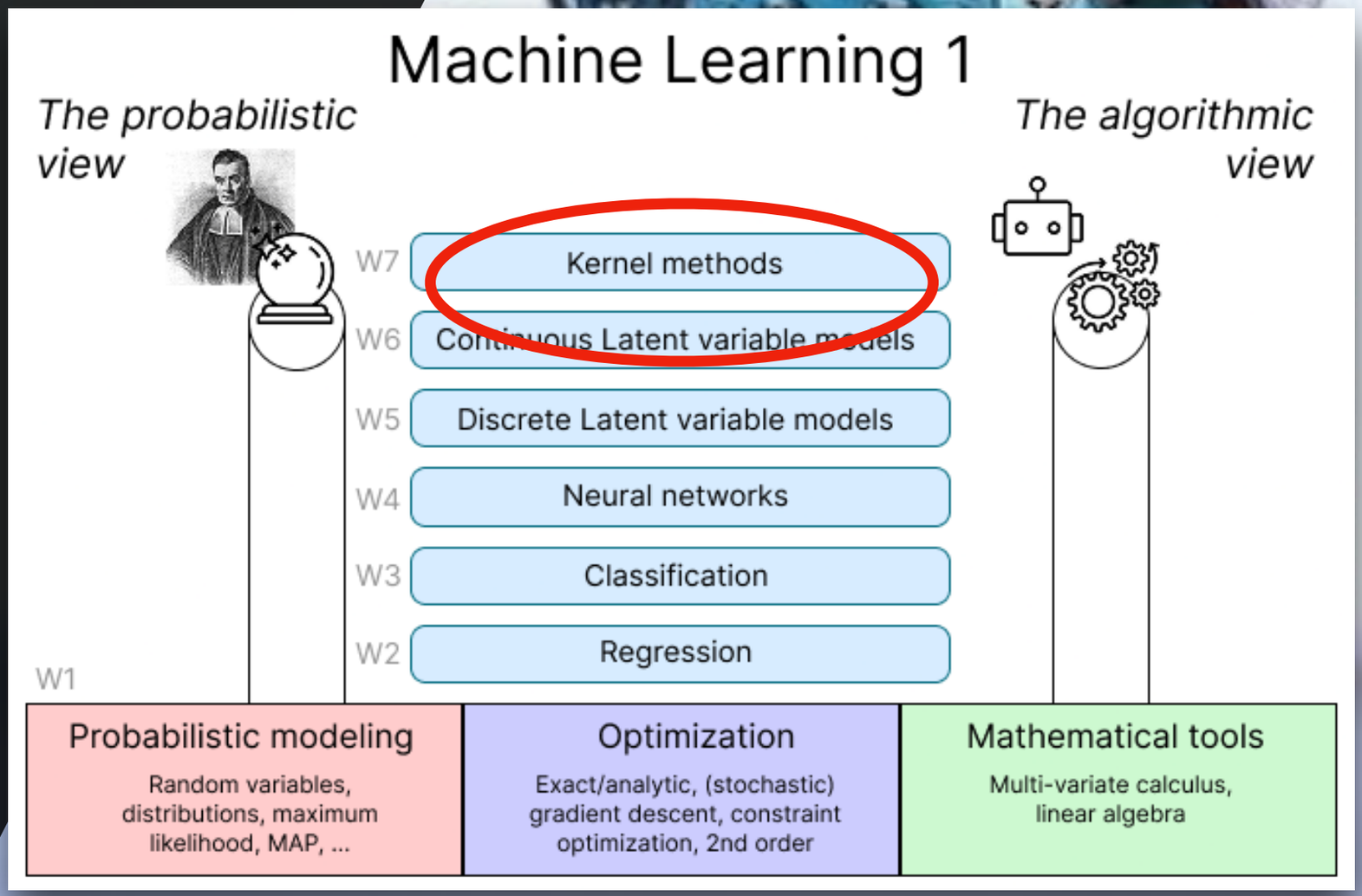
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 11.2 - Kernel Methods
Kernel Trick - Valid Kernels

Erik Bekkers

(Bishop 6.2)



Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

Kernel Trick/Kernel substitution

user specifies

What is the kernel trick?

Instead of choosing/designing/learning basis functions $\boldsymbol{\phi}(x) = \ldots$

We replace all instances of $\boldsymbol{\phi}(x)^T\boldsymbol{\phi}(y)$ with a chosen kernel $k(x, y) = \ldots$

Kernel Trick/Kernel substitution

What is the kernel trick?

Instead of choosing/designing/learning basis functions $\boldsymbol{\phi}(x) = \ldots$

We replace all instances of $\boldsymbol{\phi}(x)^T\boldsymbol{\phi}(y)$ with a chosen kernel $k(x, y) = \ldots$

Why the kernel trick?

Because a kernel, like $k(x, y) = e^{-\frac{1}{2}(x-y)^2}$

Could equivalently be defined by $k(x, y) = \boldsymbol{\phi}(x)^T\boldsymbol{\phi}(y)$

With infinite-dimensional basis vectors!!!

I.e., with $\boldsymbol{\phi}(x) \in \mathbb{R}^\infty$ without actually having to compute them!

Note on infinite dimensional feature space of Gaussian kernels

Using the definition of $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ we can show that an exponential kernel $k(x,y) = e^{-\frac{1}{2}(x-y)^2}$ implicitly uses infinite dimensional feature vectors $\boldsymbol{\phi}(x) \in \mathbb{R}^\infty$!

Proof:

$$k(x,y) = e^{-\frac{1}{2}(x-y)^2} = \underbrace{e^{-\frac{1}{2}x^2}}_{c(x)} \underbrace{e^{-\frac{1}{2}y^2}}_{c(y)} e^{xy} = c(x)c(y) \sum_{k=0}^{\infty} \frac{(xy)^k}{k!}$$

$$= \sum_{k=0}^{\infty} \frac{c(x)x^k}{\sqrt{k!}} \frac{c(y)y^k}{\sqrt{k!}}$$

$$= \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(y)$$

with $\boldsymbol{\phi}(x) = c(x) \begin{pmatrix} 1 & x & \frac{x^2}{\sqrt{2}} & \frac{x^3}{\sqrt{3!}} & \frac{x^4}{\sqrt{4!}} & \cdots \end{pmatrix}^T \in \mathbb{R}^\infty$

Parametric: make predictions with *explicit* weights and basis functions

$$y(x', \mathbf{w}) = \boldsymbol{\phi}(x')^T \mathbf{w}$$

Non-parametric: make predictions using data points and a kernel

$$y(x', \mathbf{a}) = \sum_{n=1}^{D} a_n k(x_n, x')$$

with $k(x_n, x')$ *implicitly* defined by basis functions

Kernel Trick/Kernel substitution

- Formulate your optimization problem in such a way that the input vectors $\mathbf{x}_n$ enter only in the form of scalar products:

$$\mathbf{x}_n^T \mathbf{x}_m$$ (or when using basis functions $\boldsymbol{\phi}(\mathbf{x}_n)^T \boldsymbol{\phi}(\mathbf{x}_m)$)

- **Kernel trick**: Replace all instances of $\mathbf{x}_n^T \mathbf{x}_m$ with a kernel function

$$k(\mathbf{x}_n, \mathbf{x}_m) = K_{nm}$$

- Kernel $k(\mathbf{x}_n, \mathbf{x}_m)$ corresponds to a scalar product in some (possibly infinite dimensional) feature space.

- **Valid kernel**: Gram Matrix $\mathbf{K}$ must be symmetric positive semi definite for all possible choices of $\{\mathbf{x}_n\}_{n=1}^N$

Kernels obtained from explicit basis functions are PSD.
To proof $\forall_{\mathbf{z}\in\mathbb{R}^N}: \ \mathbf{z}^T\mathbf{K}\mathbf{z} \geq 0$ with $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$.
Proof: $\mathbf{z}^T\mathbf{K}\mathbf{z} = \mathbf{z}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{z} = (\boldsymbol{\Phi}\mathbf{z})^T(\boldsymbol{\Phi}\mathbf{z}) = \|\boldsymbol{\Phi}\mathbf{z}\|^2 \geq 0$

Deriving the corresponding feature vector

- **Theorem:** For every positive definite kernel $k(\mathbf{x}, \mathbf{x}')$ there exists $\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^M$ such that

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x})$$

- In other words. Pick any *valid* kernel $k(\mathbf{x}, \mathbf{x}') = \ldots$, then it can be written in terms of some basis functions via $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x})$

- Depending on the kernel, M can be infinite!

- In general difficult to retrieve the corresponding $\boldsymbol{\phi}(\mathbf{x})$ for a given kernel

Example: polynomial kernel

- Polynomial kernel of order 2 for $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$

$$k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T\mathbf{z})^2 = (1 + x_1z_1 + x_2z_2)^2$$

$$= 1 + 2x_1z_1 + 2x_2z_2 + (x_1z_1)^2 + (x_2z_2)^2 + 2x_1z_1x_2z_2$$

$$= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)^T(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

$$= \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{z})$$

- And thus the corresponding kernel $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^6$

Constructing valid kernels

- Construct kernel from explicit set of basis functions:

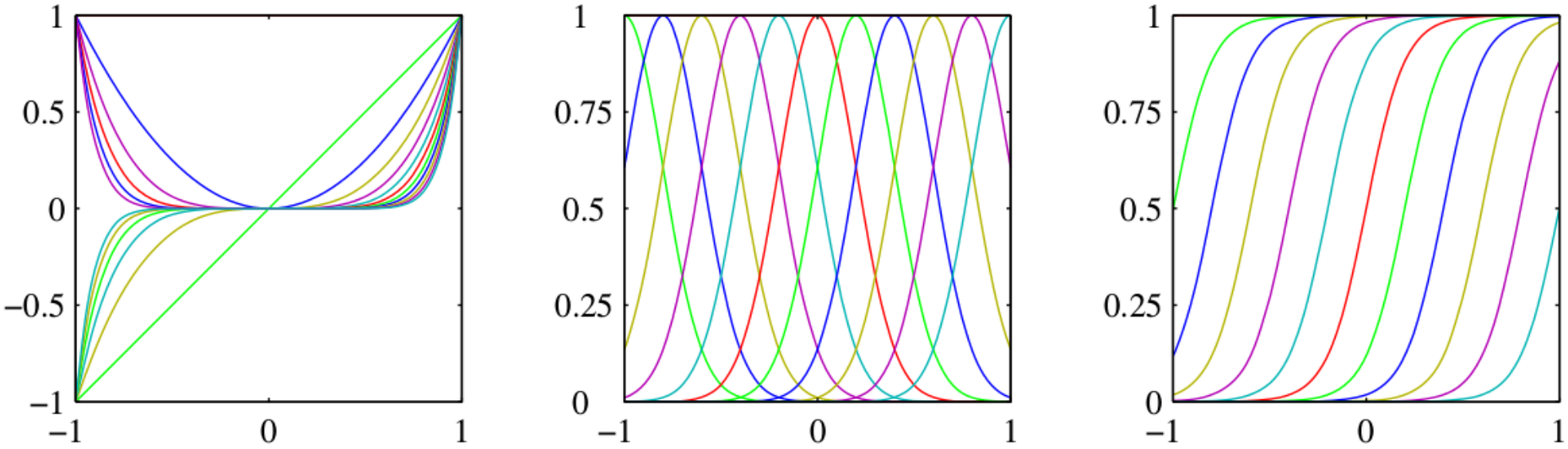
$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

$$\begin{pmatrix} \boldsymbol{\Sigma} = \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U} \\ \boldsymbol{\Sigma}^{1/2} = \mathbf{U}^T \boldsymbol{\Lambda}^{1/2} \mathbf{U} \\ \boldsymbol{\psi}(\mathbf{x}) = \boldsymbol{\Sigma}^{1/2} \boldsymbol{\phi}(\mathbf{x}) \end{pmatrix}$$

Recall lecture 5.1 (equivalent kernel)

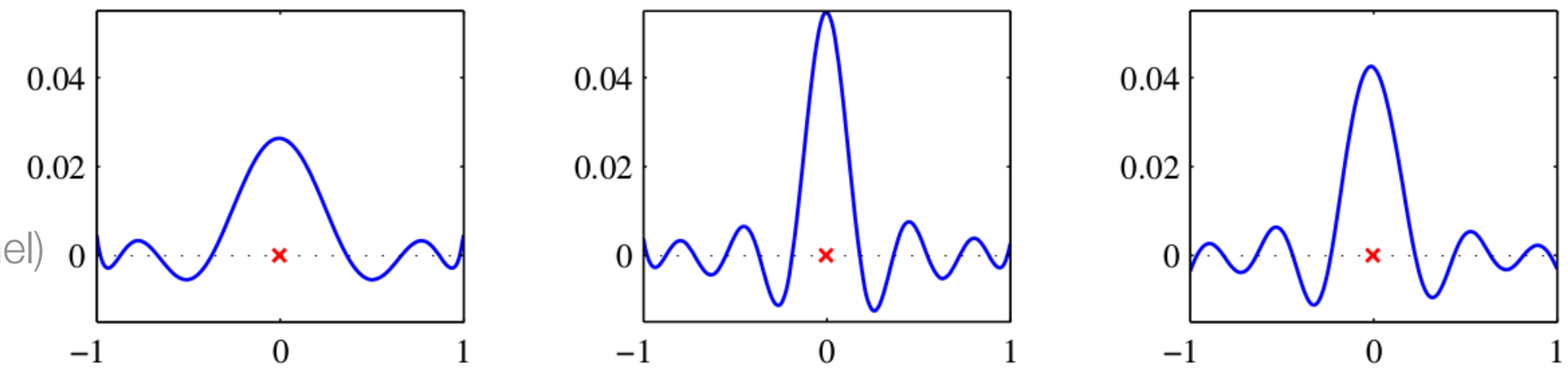
$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\Sigma} \boldsymbol{\phi}(\mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})^T \boldsymbol{\psi}(\mathbf{x}') = \sum_{i=1}^{M} \psi_i(\mathbf{x}) \psi_i(\mathbf{x}')$$

$\phi_i(\mathbf{x})$



$k(\mathbf{x},0)$

(equivalent kernel)



Examples of valid kernels

- Generalized polynomial kernel

$$k(\mathbf{x}, \mathbf{x}') = (c + \mathbf{x}^T \mathbf{x}')^M$$

- Gaussian kernel/squared exponential kernel: infinite dimensional space!

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{x}'||^2)$$

Gaussian is a type of radial basis function

- Radial basis functions:

$$k(\mathbf{x}, \mathbf{x}') = k(||\mathbf{x} - \mathbf{x}'||^2)$$

↳ depend on distance between

Construct new kernels from other kernels

Bishop page 296:

$$
\begin{aligned}
k(\mathbf{x},\mathbf{x}') &= ck_1(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= q\left(k_1(\mathbf{x},\mathbf{x}')\right) \\
k(\mathbf{x},\mathbf{x}') &= \exp\left(k_1(\mathbf{x},\mathbf{x}')\right) \\
k(\mathbf{x},\mathbf{x}') &= k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}') \\
k(\mathbf{x},\mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}),\boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x},\mathbf{x}') &= \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \\
k(\mathbf{x},\mathbf{x}') &= k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b) \\
k(\mathbf{x},\mathbf{x}') &= k_a(\mathbf{x}_a,\mathbf{x}'_a)k_b(\mathbf{x}_b,\mathbf{x}'_b)
\end{aligned}
$$

See example Bishop 6.2:

A Gaussian kernel

$$k(\mathbf{x},\mathbf{x}') = \exp(-\tfrac{1}{2}\|\mathbf{x}-\mathbf{x}'\|^2)$$

can be derived from just a linear kernel:

$$k(\mathbf{x},\mathbf{x}') = \mathbf{x}^T\mathbf{x}'$$

University of Amsterdam
Informatics Institute

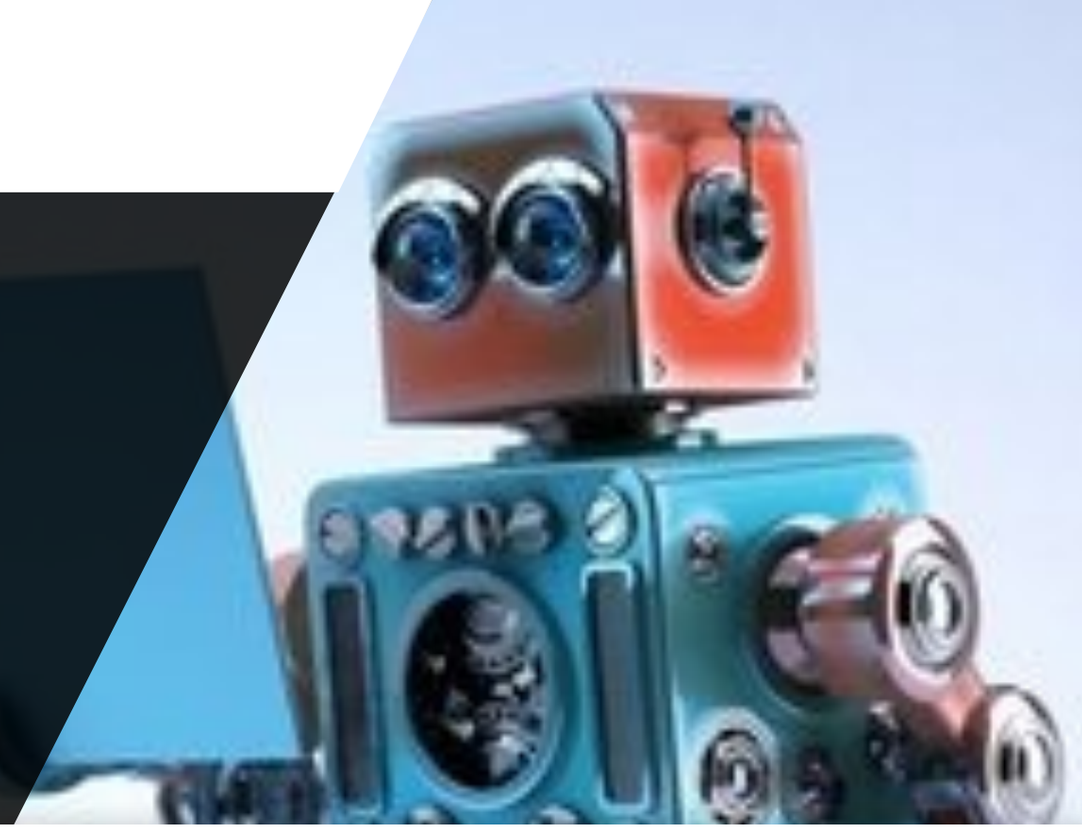
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 11.3 - Kernel Methods
Support Vector Machines - Maximum Margin Classifier

Erik Bekkers

(Bishop 7.1.0)

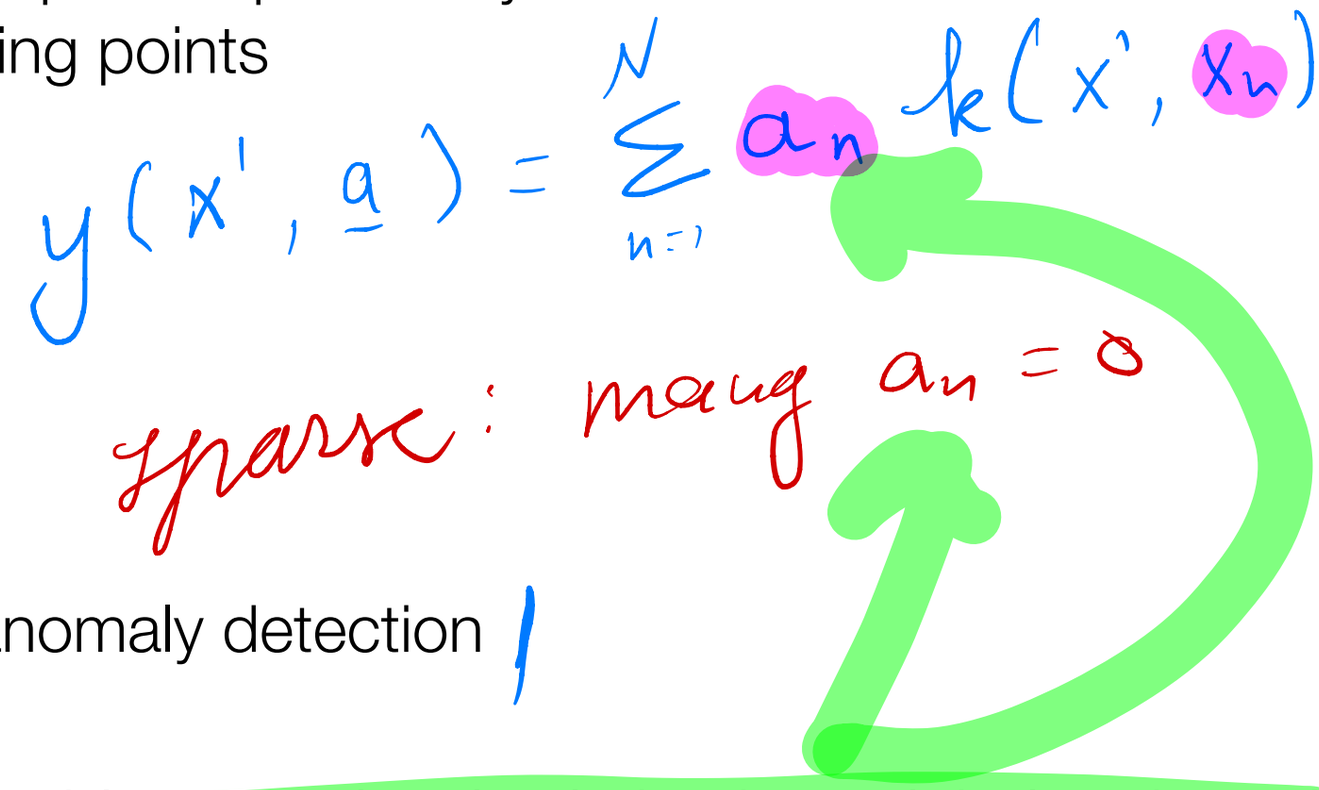


Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

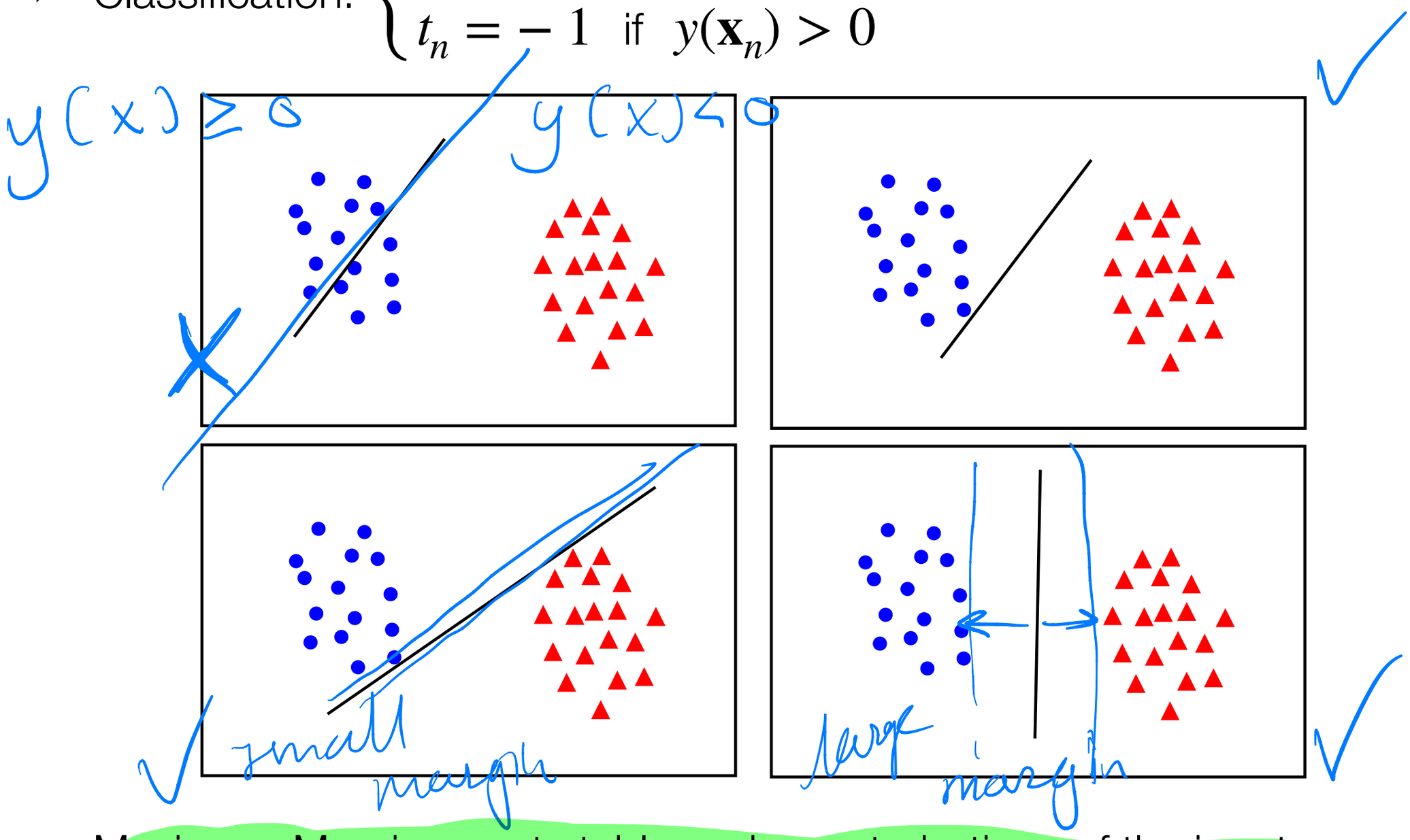
Support vector machines

- Kernel method with sparse solutions:
  - prediction for new inputs depend only on kernel function evaluated at a **subset** of the training points
- Applications:
  - Classification
  - Regression
  - novelty detection/anomaly detection
- Convex optimization problem, any local solution is at global optimum!
- No good probabilistic interpretation
- Today: SVM for binary classification -> maximum margin classifier!



Linearly separable dataset

- Linear classifier: $y(\mathbf{x_n}) = \mathbf{w}^t\mathbf{x}_n + b$
- Classification: $\begin{cases} t_n = +1 \text{ if } y(\mathbf{x}_n) > 0 \\ t_n = -1 \text{ if } y(\mathbf{x}_n) > 0 \end{cases}$



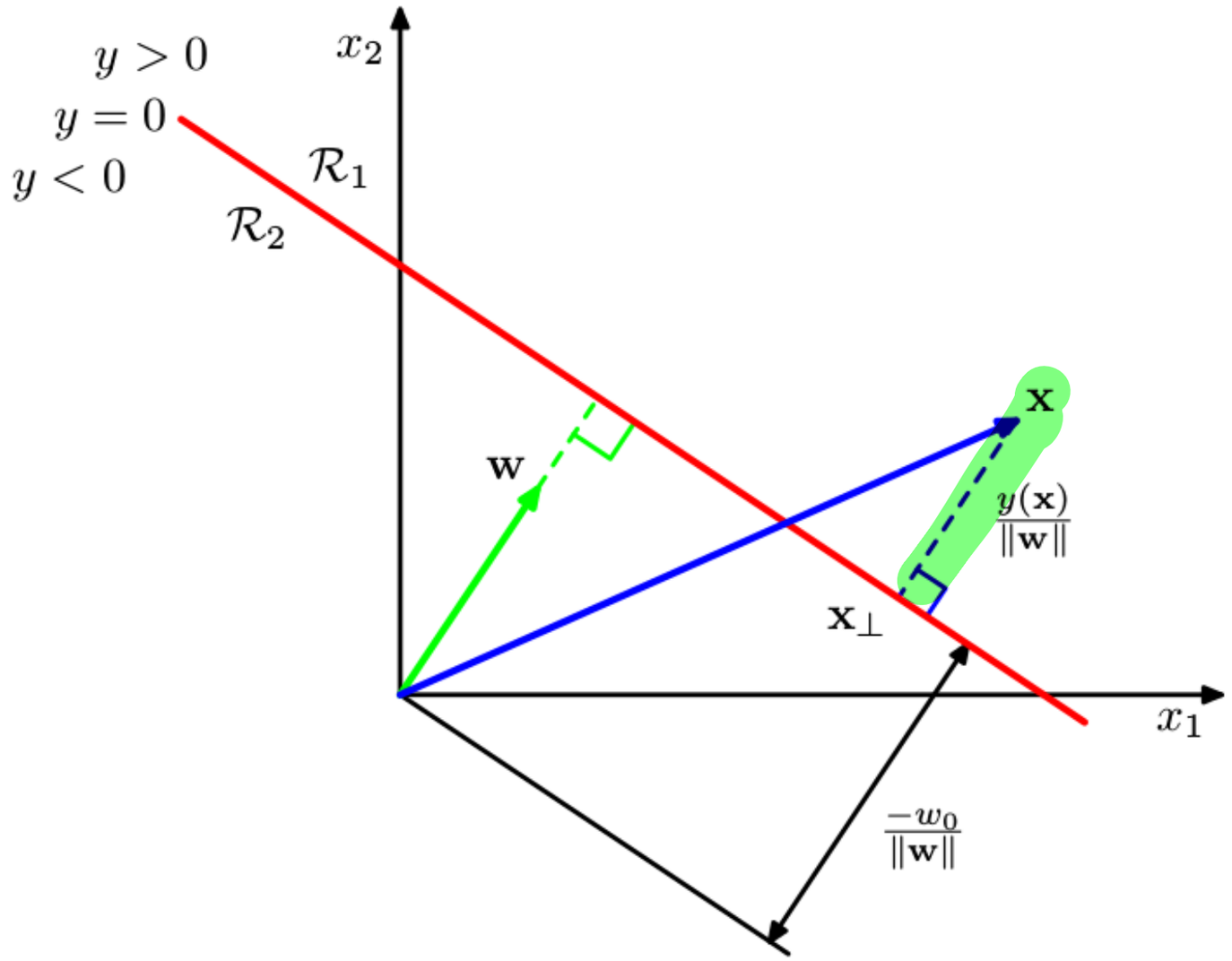
- Maximum Margin: most stable under perturbations of the input

Linearly Separable Dataset

- If $\mathbf{x}'$ lies on decision boundary: $y(\mathbf{x}') = \mathbf{w}^T\mathbf{x}' + b = 0$

- Recall: distance from $\mathbf{x}$ to decision boundary is

$$r = \frac{|y(\mathbf{x}_n)|}{\|\mathbf{w}\|} = \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|}$$



- For correct classification:

$$y(\mathbf{x}_n) > 0 \text{ if } t_n = +1$$

$$y(\mathbf{x}_n) < 0 \text{ if } t_n = -1$$

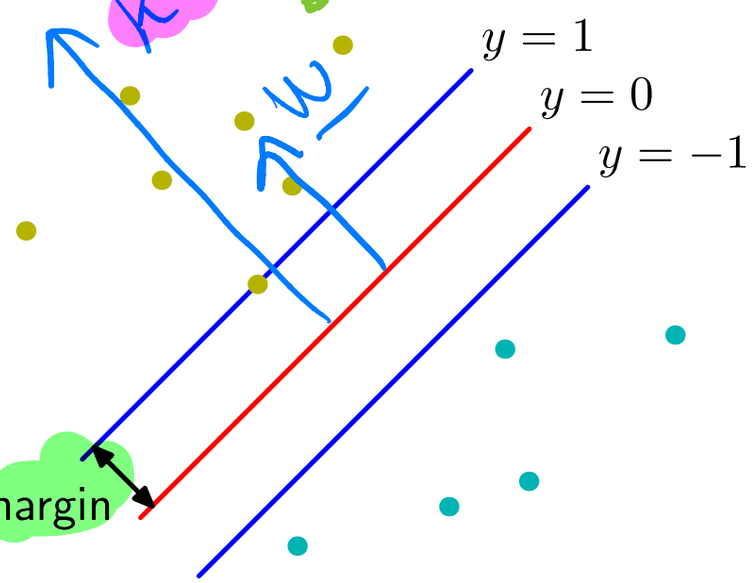
- So for all $n = 1, \ldots, N$

$$t_n y(\mathbf{x}_n) > 0$$

Maximum Margin Classifier

- Margin: perpendicular distance from decision boundary to closest point $\mathbf{x}_n$

define the same margin



- For all data points distance to decision boundary is

$$r = \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|\mathbf{w}\|}$$

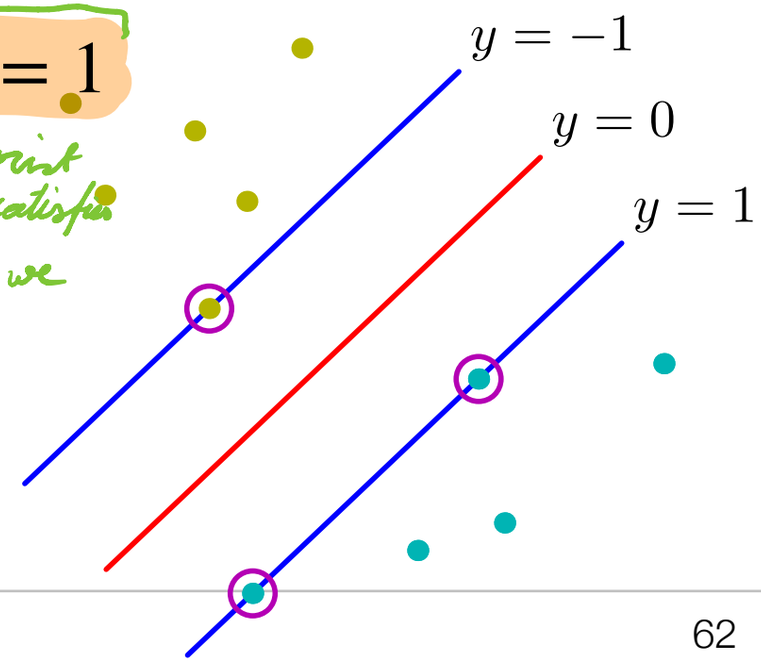
- Margin: $\min_n \frac{t_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|\mathbf{w}\|} = \min_n \frac{t_n(\kappa\mathbf{w}^T\mathbf{x}_n + \kappa b)}{\|\kappa\mathbf{w}\|}$

our choice

- For point closest to decision boundary $t_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$

we can always "recalibrate" using a k s.t. $t_n(\underline{w}^T x_n + b) = 1$ for the closest point so let's just decide that we are only interested in directly finding the $\underline{w}$ that satisfies this constraint. Thus we consider

- For all data points: $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$



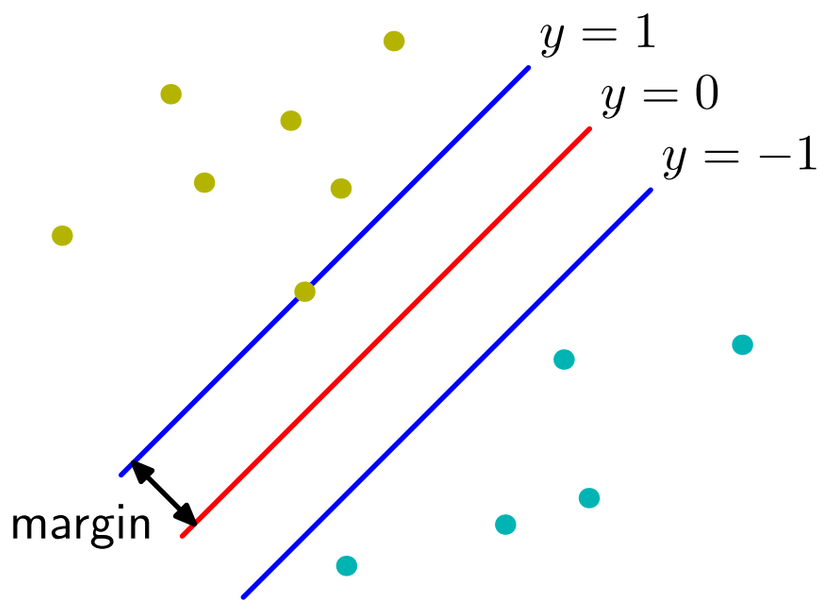
- Maximum margin classifier:

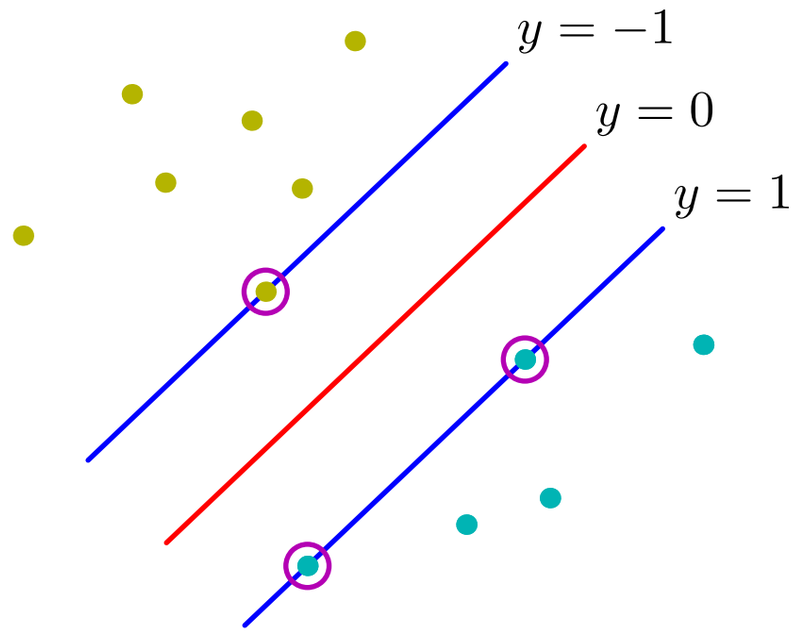
Maximum Margin Classifier

- For all data points distance to decision boundary is

$$r = \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|\mathbf{w}\|}$$

- For point closest to decision boundary $t_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$
- For all data points: $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
- Size of the margin: $\frac{1}{\|\mathbf{w}\|}$





Maximum Margin Classifier

- Size of the margin: $\frac{1}{\|\mathbf{w}\|}$

$\underline{w}^* = \text{argmax}\left[\frac{1}{\|\mathbf{w}\|}\right] \Longleftrightarrow \text{argmin} \|\underline{w}\|^2$

- For all data points: $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
- **Maximizing the margin**:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \textit{ subject to } N \textit{ constraints } t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$$

- Quadratic programming problem!