Kernels

Week 4a - Statistical ML / Thang Bui / ANU / 2025 S1

Housekeeping

+ Assignment 1 due in <3 weeks [mid-night Fri 28/3]. 5% penalty for 5 mins - 24 hrs late, 100% penalty after.
+ Wattle quiz will be released on Monday, available for 3 days, one attempt only.
+ Feedback: please email us or contact your course reps.

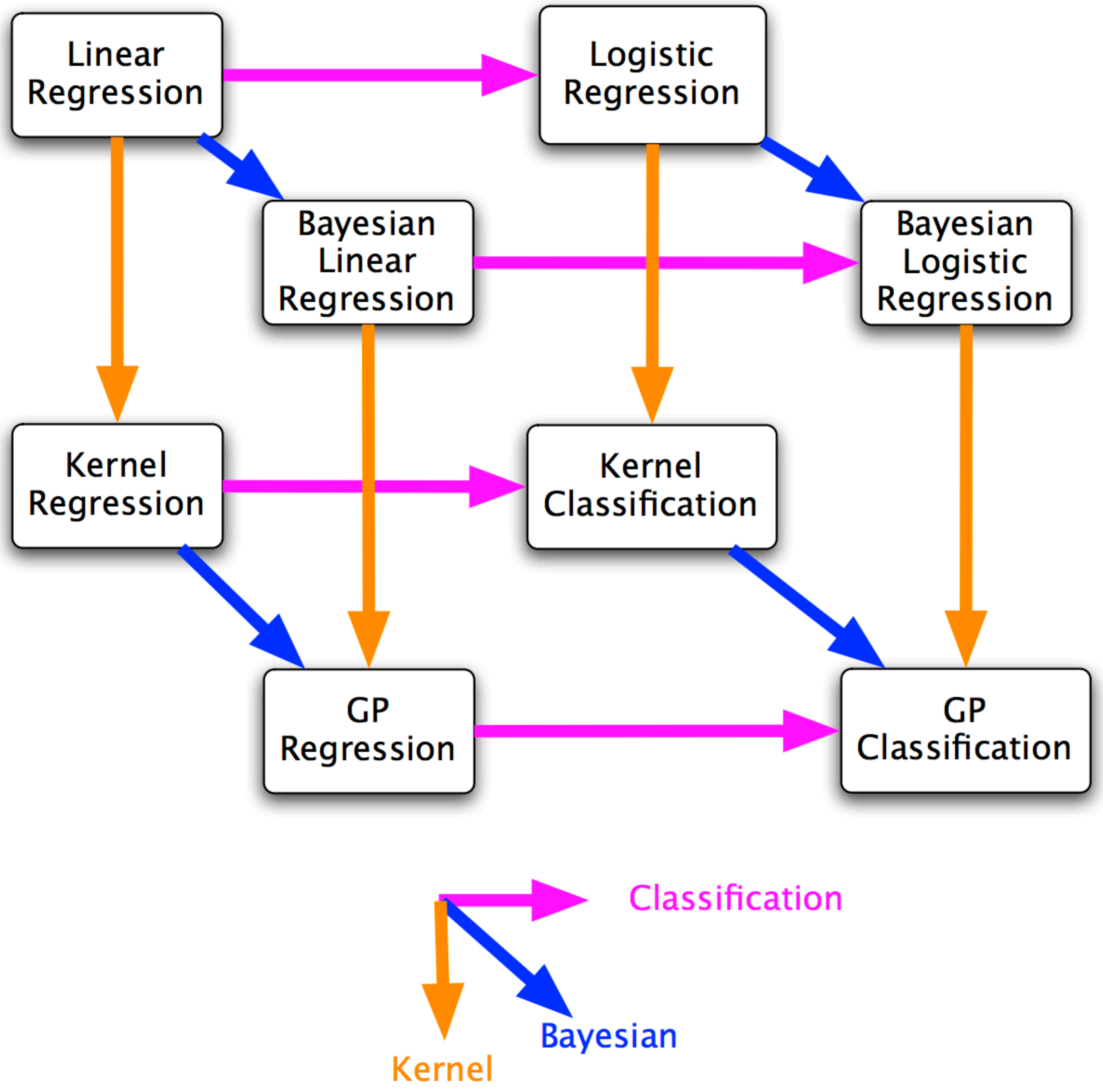
Plan for next few weeks

Week	Mon lecture	Wed lecture	Tutorial
Week 1	Intro	Background	
Week 2	Linear regression	Bayesian Linear Regression	Regression
Week 3	Logistic regression	Classification, Bayesian logistic reg	
Week 4	Kernels	GP regression	Classification
Week 5	GP regression	GP classification	
Week 6	GP classification	GP approximations + recap	Kernels

Overview

1. Revisit regularised linear regression
2. Kernels: definition and construction
3. Kernel logistic classification

Reading: Bishop 6.1, 6.2



Linear regression example

Linear regression example

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Linear regression example

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

+ Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$
+ Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Linear regression example

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

+ Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$
+ Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Test time: given a new input $\mathbf{x}^*$, prediction = $f(\mathbf{x}^*) = \theta^\intercal \mathbf{x}^*$

Linear regression example

Training data N input, output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R}$

Each row is a data point

Each column is a feature dimension

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One scalar per data point

Assumptions:

+ Underlying function is **linear**, $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

+ Due to measurement noise, observed y is a noisy version of $f(\mathbf{x})$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Test time: given a new input $\mathbf{x}^*$, prediction = $f(\mathbf{x}^*) = \theta^\intercal \mathbf{x}^*$

Whiteboard: closed form gradients, optimal parameters, and predictions

Linear regression - summary

Primal	Dual
parameters θ	weights α with $\theta = \sum_n \alpha_n \phi_n$
optimal $\theta = (\Phi^\intercal \Phi + \lambda \mathrm{I}_D)^{-1} \Phi^\intercal \mathbf{y}$	optimal $\alpha = (\Phi \Phi^\intercal + \lambda \mathrm{I}_N)^{-1} \mathbf{y}$
prediction $f(x_*) = \phi_*^\intercal \theta$	prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$
complexity $\mathcal{O}(D^3 + D^2 N)$	complexity $\mathcal{O}(N^3 + N^2 D)$

Linear regression - summary

Primal	Dual
parameters θ	weights α with $\theta = \sum_n \alpha_n \phi_n$
optimal $\theta = (\Phi^\intercal \Phi + \lambda \mathrm{I}_D)^{-1} \Phi^\intercal \mathbf{y}$	optimal $\alpha = (\Phi \Phi^\intercal + \lambda \mathrm{I}_N)^{-1} \mathbf{y}$
prediction $f(x_*) = \phi_*^\intercal \theta$	prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$
complexity $\mathcal{O}(D^3 + D^2 N)$	complexity $\mathcal{O}(N^3 + N^2 D)$

We prefer the dual formulation when D > N.
The dual also allows using high-dimensional (even infinite) features without computing them!
KERNEL TRICK!

Kernelisation or the kernel trick

Kernelisation or the kernel trick

Dual parameterisation

weights α with $\theta = \sum_n \alpha_n \phi_n$

optimal $\alpha = (\Phi\Phi^\intercal + \lambda \mathrm{I}_N)^{-1}\mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$

complexity $\mathcal{O}(N^3 + N^2 D)$

Kernelisation or the kernel trick

kernel function $k(x, z) = \phi(x)^\intercal \phi(z)$

Gram matrix, $N \times N$, K where $K_{ij} = \phi(x_i)^\intercal \phi(x_j)$

optimal $\alpha = (K + \lambda \mathrm{I}_N)^{-1} \mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n k(x_n, x_*)$

Dual parameterisation

weights α with $\theta = \sum_n \alpha_n \phi_n$

optimal $\alpha = (\Phi \Phi^\intercal + \lambda \mathrm{I}_N)^{-1} \mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$

complexity $\mathcal{O}(N^3 + N^2 D)$

Kernelisation or the kernel trick

Dual parameterisation

kernel function $k(x, z) = \phi(x)^\intercal \phi(z)$

Gram matrix, $N \times N$, K where $K_{ij} = \phi(x_i)^\intercal \phi(x_j)$

optimal $\alpha = (K + \lambda \mathrm{I}_N)^{-1}\mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n k(x_n, x_*)$

weights α with $\theta = \sum_n \alpha_n \phi_n$

optimal $\alpha = (\Phi\Phi^\intercal + \lambda \mathrm{I}_N)^{-1}\mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$

complexity $\mathcal{O}(N^3 + N^2 D)$

But hold on, what is the trick, we still need to compute phi first and then compute k?

Kernelisation or the kernel trick

kernel function $k(x, z) = \phi(x)^\intercal \phi(z)$

Gram matrix, $N \times N$, K where $K_{ij} = \phi(x_i)^\intercal \phi(x_j)$

optimal $\alpha = (K + \lambda \mathrm{I}_N)^{-1}\mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n k(x_n, x_*)$

Dual parameterisation

weights α with $\theta = \sum_n \alpha_n \phi_n$

optimal $\alpha = (\Phi\Phi^\intercal + \lambda \mathrm{I}_N)^{-1}\mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n \phi_n^\intercal \phi_*$

complexity $\mathcal{O}(N^3 + N^2 D)$

But hold on, what is the trick, we still need to compute phi first and then compute k?

Example: $k(x, z) = (1 + x^\intercal z)^p$ is a polynomial kernel, equivalent to having many polynomial features (up to order p). And we don't need to compute these $\mathcal{O}(D^p)$ features explicitly!

Kernel: properties and construction

kernel function $k(x, z) = \phi(x)^\intercal \phi(z)$

Symmetric and positive semidefinite
Show these!

Gram matrix, $N \times N$, K where $K_{ij} = \phi(x_i)^\intercal \phi(x_j)$

Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Kernel: examples

Further examples of kernels

$$k(\mathbf{x},\mathbf{x}') = (\mathbf{x}^\top\mathbf{x}')^M$$ only terms of degree M

$$k(\mathbf{x},\mathbf{x}') = (\mathbf{x}^\top\mathbf{x}' + c)^M$$ all terms up to degree M

$$k(\mathbf{x},\mathbf{x}') = \exp\left(-\|\mathbf{x}-\mathbf{x}'\|^2/2\sigma^2\right)$$ Gaussian kernel

$$k(\mathbf{x},\mathbf{x}') = \tanh\left(a\,\mathbf{x}^\top\mathbf{x}' + b\right)$$ Sigmoidal kernel (invalid)

Show that this is a valid kernel
Also called Radial Basis Function (RBF),
Squared exponential, or exponentiated quadratic

$$\Phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)\left[1, \frac{x}{\sigma\sqrt{1!}}, \frac{x^2}{\sigma^2\sqrt{2!}}, \frac{x^3}{\sigma^3\sqrt{3!}}, \ldots\right]^\top$$

https://www.cs.toronto.edu/~duvenaud/cookbook/

Generally, we call

$$k(\mathbf{x},\mathbf{x}') = \mathbf{x}^\top\mathbf{x}'$$ linear kernel

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

$$k(\mathbf{x},\mathbf{x}') = k(\|\mathbf{x}-\mathbf{x}'\|)$$ homogeneous kernel

Kernel: more examples

Kernels over sets, graphs, strings

- We 'only' need an appropriate similarity measure $k(\mathbf{x}, \mathbf{x}')$ which is a kernel.
- Example: Given a set $\mathcal{A}$ and the set of all subsets of $\mathcal{A}$, called the power set $\mathcal{P}(\mathcal{A})$.
- For two subsets $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{P}(\mathcal{A})$, denote the number of elements of the intersection of $\mathcal{A}_1$ and $\mathcal{A}_2$ by $|\mathcal{A}_1 \cap \mathcal{A}_2|$.
- Then it can be shown that

$$k(\mathcal{A}_1, \mathcal{A}_2) = 2^{|\mathcal{A}_1 \cap \mathcal{A}_2|}$$

  corresponds to an inner product in a feature space. Therefore, $k(\mathcal{A}_1, \mathcal{A}_2)$ is a valid kernel function.

Kernels from probabilistic generative models

- Given $p(\mathbf{x})$, we can define a kernel

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

  which means two inputs $\mathbf{x}$ and $\mathbf{x}'$ are similar if they both have high probabilities.
- Include a weighting function $p(i)$ and extend the kernel to

$$k(\mathbf{x}, \mathbf{x}') = \sum_i p(\mathbf{x} \mid i)\, p(\mathbf{x}' \mid i) p(i).$$

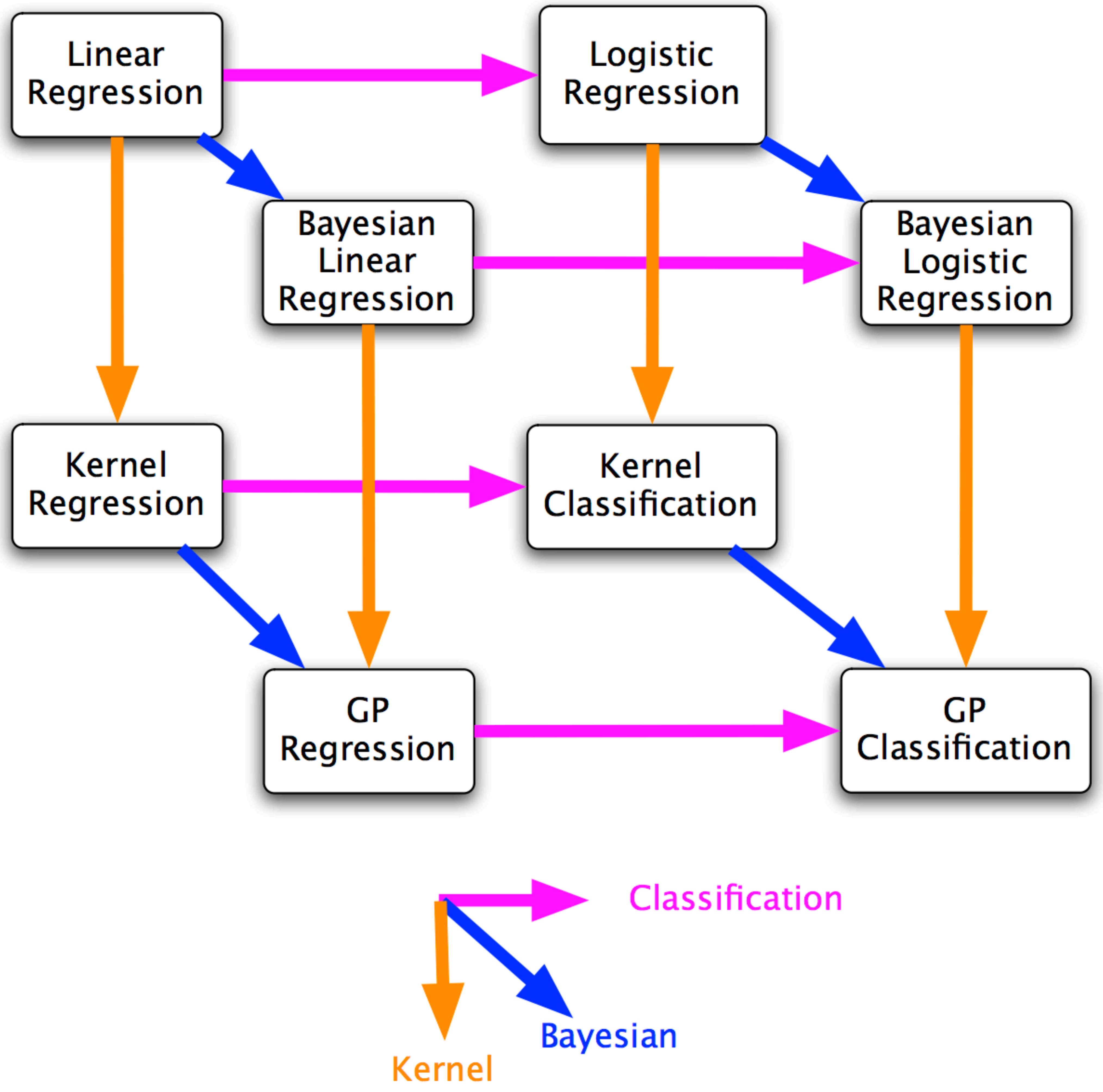
- For a continous variable $\mathbf{z}$

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{x}' \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

Overview

1. Revisit regularised linear regression
2. Kernels: definition and construction
3. **Kernel logistic classification**

Reading: Bishop 6.1, 6.2



Binary logistic classification - problem set-up

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,$ $y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \mid x)$

For binary classification, by laws of probabilities: $p(y = 0 \mid x) + p(y = 1 \mid x) = ?$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = \,?$ 1

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,\ y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = \,?$ 1

Imagine: $p(y = 1 \,|\, x) = g_\theta(x)$ then $p(y = 0 \,|\, x) = 1 - g_\theta(x)$. **Constraint**: $0 \leq g_\theta(x) \leq 1, \forall x$

Binary logistic classification - problem set-up

Training data N input, output pairs $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \mathbf{x}_n \in \mathbb{R}^D,$ $y_n \in \{0,1\}$

Each row is a data

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

One value per data point
0 or 1

We construct a **classifier** that learns the class probabilities: $p(y = c \,|\, x)$

For binary classification, by laws of probabilities: $p(y = 0 \,|\, x) + p(y = 1 \,|\, x) = \,?$ 1

Imagine: $p(y = 1 \,|\, x) = g_\theta(x)$ then $p(y = 0 \,|\, x) = 1 - g_\theta(x)$. **Constraint**: $0 \leq g_\theta(x) \leq 1, \forall x$

Prediction with threshold = 0.5: $y = 1$ if $g_\theta(x) \geq 0.5$ and $y = 0$ if $g_\theta(x) < 0.5$.

Binary logistic classification - logistic function

Binary logistic classification - logistic function

Reminder: Linear regression: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Binary logistic classification - logistic function

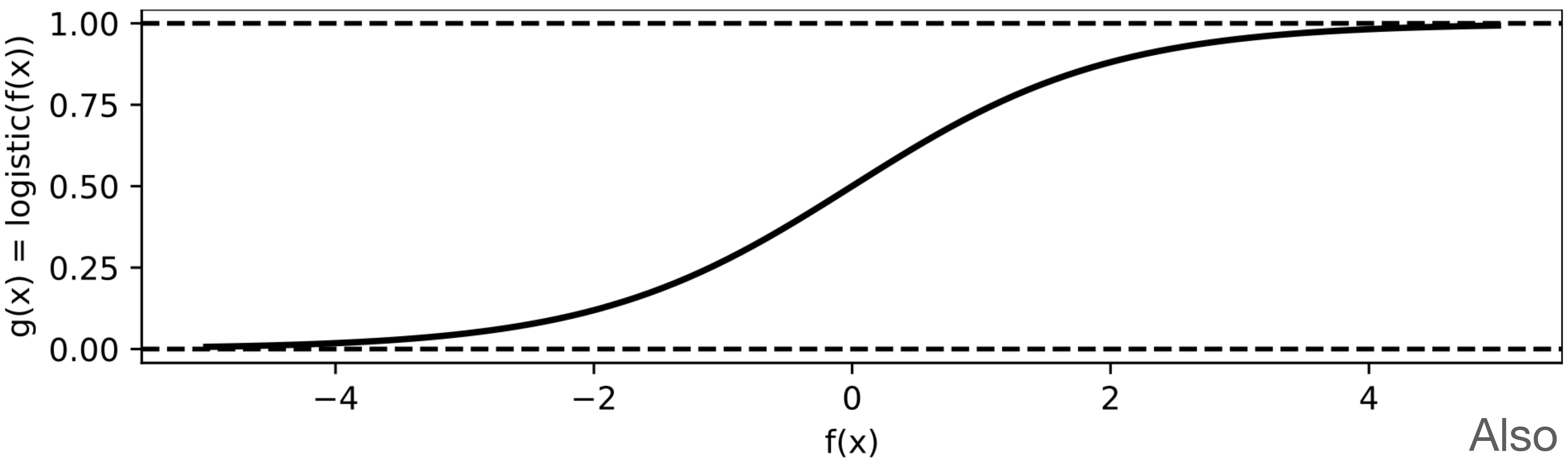
Reminder: Linear regression: $f_\theta(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

$$\begin{bmatrix} f_\theta(\mathbf{x}_1) \\ f_\theta(\mathbf{x}_2) \\ \vdots \\ f_\theta(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \theta^\intercal \mathbf{x}_1 \\ \theta^\intercal \mathbf{x}_2 \\ \vdots \\ \theta^\intercal \mathbf{x}_N \end{bmatrix} = X\theta$$

Want: $0 \leq g_\theta(x) \leq 1, \forall x$

Idea: 'squash' $f_\theta(x)$ through a *logistic sigmoid* function $g_\theta(x) = \sigma(f_\theta(x))$, where

$$\sigma(z) = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$



Also called sigmoid

Binary logistic classification - maximum likelihood

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n | x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n \mid x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

We want to *maximise* the likelihood or *minimise* the negative log-likelihood:

$$\mathcal{L}_n(\theta) = -\log p(y_n \mid x_n, \theta) = \begin{cases} -\log(g_\theta(x_n)), & \text{if } y_n = 1 \\ -\log(1 - g_\theta(x_n)), & \text{if } y_n = 0 \end{cases}$$

$$= -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n))$$

Binary logistic classification - maximum likelihood

We can write down the ***likelihood*** of parameters given one data point:

$$p(y_n | x_n, \theta) = \begin{cases} g_\theta(x_n), & \text{if } y_n = 1 \\ 1 - g_\theta(x_n), & \text{if } y_n = 0 \end{cases}$$

We want to *maximise* the likelihood or *minimise* the negative log-likelihood:

$$\mathscr{L}_n(\theta) = -\log p(y_n | x_n, \theta) = \begin{cases} -\log(g_\theta(x_n)), & \text{if } y_n = 1 \\ -\log(1 - g_\theta(x_n)), & \text{if } y_n = 0 \end{cases}$$

$$= -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n))$$

For N datapoints:

$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} -y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n))$$

Often called the **binary cross-entropy loss**

Binary logistic classification - gradients

Objective: $\mathscr{L}(\theta) = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} - y_n \log(g_\theta(x_n)) - (1 - y_n)\log(1 - g_\theta(x_n))$

Notes: $g_\theta(x) = \sigma(f_\theta(x))$, σ is a logistic sigmoid, and $f_\theta(\mathbf{x}) = \displaystyle\sum_{d=1}^{D} \theta_d x_d = \theta^\intercal \mathbf{x}, \theta \in \mathbb{R}^D$

Show: $\dfrac{\mathrm{d}\mathscr{L}(\theta)}{\mathrm{d}\theta} = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} (g_\theta(x_n) - y_n)x_n^T$

How do we kernelise this?

One more thing: parametric vs nonparametric

- Parametric methods
  - Learn the model parameter $\mathbf{w}$ from the training data $\mathbf{t}$.
  - Discard the training data $\mathbf{t}$.
- Nonparametric methods
  - Use training data directly for prediction
  - k-nearest neighbours : use k-closest data from the 'training' set for classification
- Kernel methods
  - Base prediction on linear combination of kernel functions evaluated at the training data.

Is this parametric or non-parametric?

One more thing: parametric vs nonparametric

- Parametric methods
  - Learn the model parameter $\mathbf{w}$ from the training data $\mathbf{t}$.
  - Discard the training data $\mathbf{t}$.
- Nonparametric methods
  - Use training data directly for prediction
  - k-nearest neighbours : use k-closest data from the 'training' set for classification
- Kernel methods
  - Base prediction on linear combination of kernel functions evaluated at the training data.

kernel function $k(x, z) = \phi(x)^\intercal \phi(z)$

Gram matrix, $N \times N$, K where $K_{ij} = \phi(x_i)^\intercal \phi(x_j)$

optimal $\alpha = (K + \lambda \mathrm{I}_N)^{-1} \mathbf{y}$

prediction $f(x_*) = \sum_n \alpha_n k(x_n, x_*)$

Is this parametric or non-parametric?