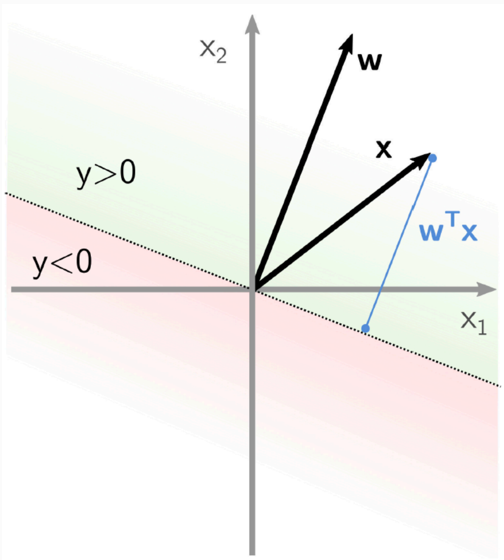
Lecture 3: Regression in Linear Models - 27/01/20

Linear Regression

A simple ***linear model*** for vector inputs $\mathbf{x} \in \mathbb{R}^{D-1}$:

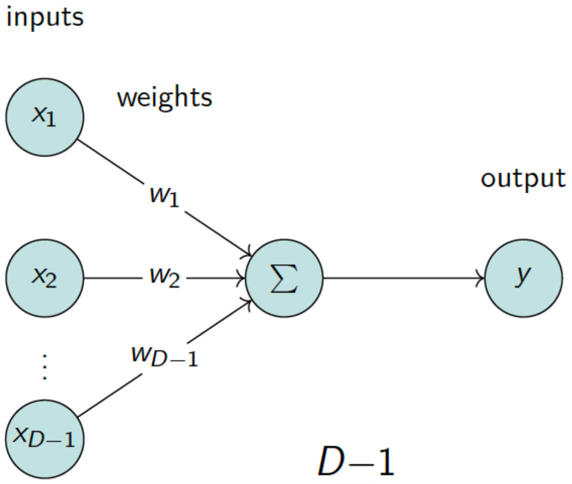
$$
\begin{aligned}
y(\mathbf{x}) &= \sum_{d=1}^{D-1} w_d x_d \\
&= \begin{pmatrix} w1 & \dots & w_{D-1} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{(D-1)} \end{pmatrix} \\
&= \begin{pmatrix} w_1 \\ \vdots \\ w_{(D-1)} \end{pmatrix}^T \begin{pmatrix} w_1 \\ \vdots \\ x_{(D-1)} \end{pmatrix} \\
&= \mathbf{w}^T \mathbf{x}
\end{aligned}
$$



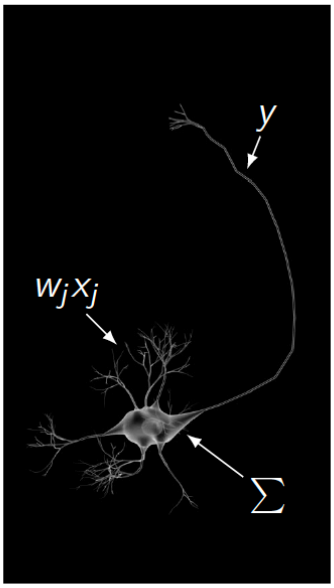
$y = 0$ is defined as the ***decision line***

A Simple Neuron

We can think of this as equivalent to a simple neuron model called the perceptron:



$$y = \sum_{d=1}^{D-1} w_d x_d$$



Bias Term

We introduce a ***bias term***:

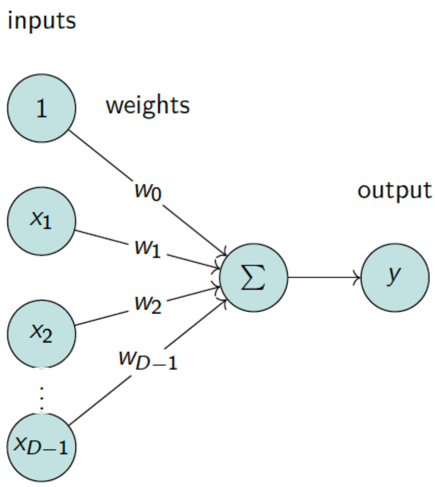
$$\begin{aligned} y(\mathbf{x}) &= w_0 + \sum_{d=1}^{D-1} w_d x_d \\ &= w_0 + \mathbf{w}^T \mathbf{x} \end{aligned}$$

For a point on the decision line:

$$\begin{aligned} y(\mathbf{x}) &= 0 \\ \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}||} &= -\frac{w_0}{||\mathbf{w}||} \end{aligned}$$

The bias can be absorbed into the vector:

$$\begin{aligned} y &= w_0 + \sum_{d=1}^{D-1} w_d x_d \\ &= \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_{(D-1)} \end{pmatrix}^T \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_{(D-1)} \end{pmatrix} \\ &= \boldsymbol{w}^T \boldsymbol{x} \end{aligned}$$



New definition of $\boldsymbol{w}$ and $\boldsymbol{x}$

Model has D parameters $\{w_0, \dots, w_{(D-1)}\}$ (degrees of freedom).

Solving the Linear Model

Each input is a vector $\mathbf{x}_n \in \mathbb{R}^D$, with corresponding target, $t \in \mathbb{R}$. We want to minimise the ***sum-of-square errors***, with the ***error function*** being:

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)^2$$

Rewrite in matrix notation:

$$E_D(\mathbf{w}) = \frac{1}{2}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})$$

with $\mathbf{t} \in \mathbb{R}^N$ is our collected targets and $(N \times D)$-matrix of inputs:

$$\mathbf{X} = \begin{pmatrix} x_{10} & x_{11} & \dots & x_{1(D-1)} \\ x_{20} & x_{21} & \dots & x_{2(D-1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N0} & x_{N1} & \dots & x_{N(D-1)} \end{pmatrix}$$

> ***Note that***: each row vector $\mathbf{x}_i^T$ is ith data input while each column vector is a set of data input $\tilde{\mathbf{x}}_j$ for jth demension

Minimise the error function $E_D(\mathbf{w})$ by differentiating and setting to zero:

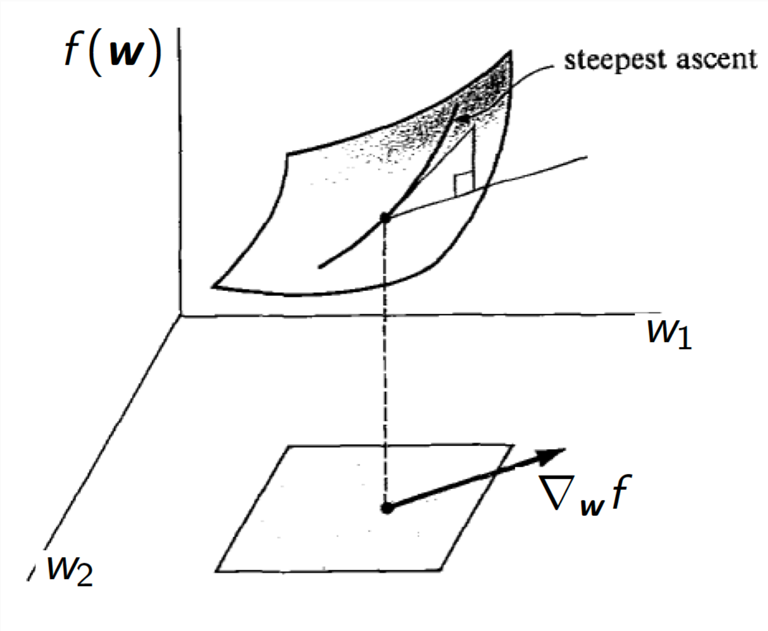
$$\begin{aligned} \nabla_{\mathbf{w}} E_D(\mathbf{w}) = \nabla_{\mathbf{w}}[\frac{1}{2}(\mathbf{t} - \mathbf{X}\mathbf{w})^T(\mathbf{t} - \mathbf{X}\mathbf{w})] = 0 \\ -\mathbf{X}^T(\mathbf{t} - \mathbf{X}\mathbf{w}^*) = 0 \end{aligned}$$

Expanding brackets, rearraging, multipling by $(\mathbf{X}^T\mathbf{X})^{-1}$

$$\begin{aligned} \mathbf{X}^T\mathbf{X}\mathbf{w}^* &= \mathbf{X}^T\mathbf{t} \\ (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\mathbf{w}^* &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} \\ \mathbf{w}^* &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} \end{aligned}$$

> ***Most Likelihood Weights Linear Regression***:
>
> $$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} = \mathbf{w}_{ML}$$

> ***The Gradient Operator*** $\nabla_{\mathbf{w}}$
>
> The gradient operator is vector of (partial) differential operations that gives direction of maximum ascent
>
> $$\nabla_{\mathbf{w}} f = \frac{df}{d\mathbf{w}} = (\frac{\delta f}{\delta w_0}, \dots, \frac{\delta f}{\delta w_{(D-1)}})^T$$



Properties:

- Gradient of dot product: $\nabla_{\mathbf{w}} \mathbf{w}^T \mathbf{v} = \mathbf{v}$
- Product rule: $\nabla_{\mathbf{x}} u(\mathbf{x}) v(\mathbf{x}) = v \nabla_{\mathbf{x}} u + u \nabla_{\mathbf{x}} v$
- Chain rule: $\nabla_{\mathbf{z}} f(g(\mathbf{z})) = \frac{df}{dg} \nabla_{\mathbf{z}} g$

The Moore-Penrose Pseudo-Inverse

$$(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{A}^\dagger \in \mathbb{R}^{M\times N}$$

$\mathbf{A}^\dagger$ is defined as the Moore-Penrose pseudo-inverse of matrix $\mathbf{A}$, which provides properties similar to the inverse of a square matrix for non-square matrix:

- Not a real inverse: $\mathbf{A}\mathbf{A}^\dagger \neq \mathbf{I}$
- Almost an inverse: $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$
- If $\mathbf{A}$ is square and invertible then $\mathbf{A}^\dagger = \mathbf{A}^{-1}$
- Can be problematic if $\mathbf{A}^T\mathbf{A}$ is (or close to) singular

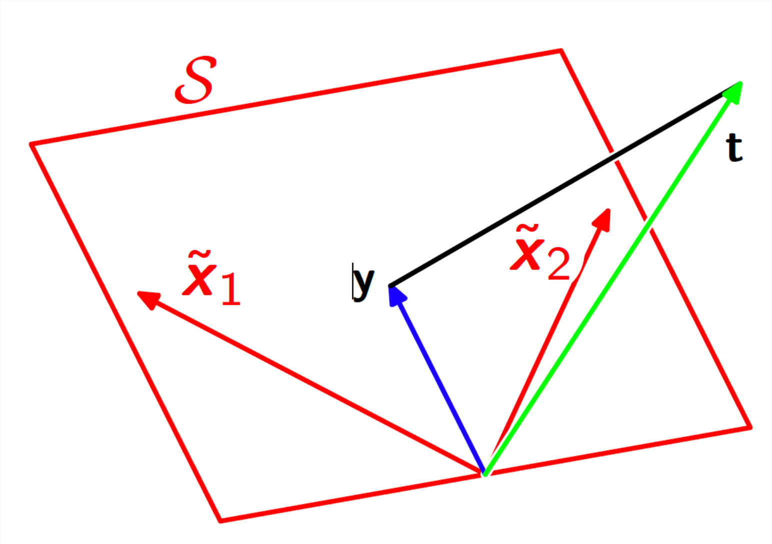
Geometric Intuition

$$\mathbf{X} = \begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1(D-1)} \\ x_{20} & x_{21} & \cdots & x_{2(D-1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N0} & x_{N1} & \cdots & x_{N(D-1)} \end{pmatrix}$$

- Row Vector $\mathbf{x}_i^T$
- Column Vector $\tilde{\mathbf{x}}_j \in \mathbb{R}^N$
- $\mathbf{t}$ is a vector in $\mathbb{R}^N$
- S is (sub-)space spanned by $\{\tilde{\mathbf{x}}_d\}$
- $\dim(S) \leq D$

  > Some of the data input might not be linearly independent

- $\mathbf{y} = \mathbf{X}\mathbf{w}^*$ is point in S closest to $\mathbf{t}$



kNN for Regression

k-Nearest Neighbours (***kNN***) assumes estimates $y(\mathbf{x})\&y(\mathbf{x}')$ are similar, when $\mathbf{x}$ is close to $\mathbf{x}'$:

Predicts:

$$y(\mathbf{x}, k) = \frac{1}{k} \sum_{\mathbf{x}_i \in \mathbb{N}_k(\mathbf{x})} t_i$$

when $\mathbb{N}_k\mathbf{x}$ contains the k closest points to $\mathbf{x}$

> In words, ***predict for $y(\mathbf{x})$ the average target of the k nearest points***

- A closeness measure, e.g. Euclidean distance, is required
- Usualy more common for ***classification***

Pseudocode

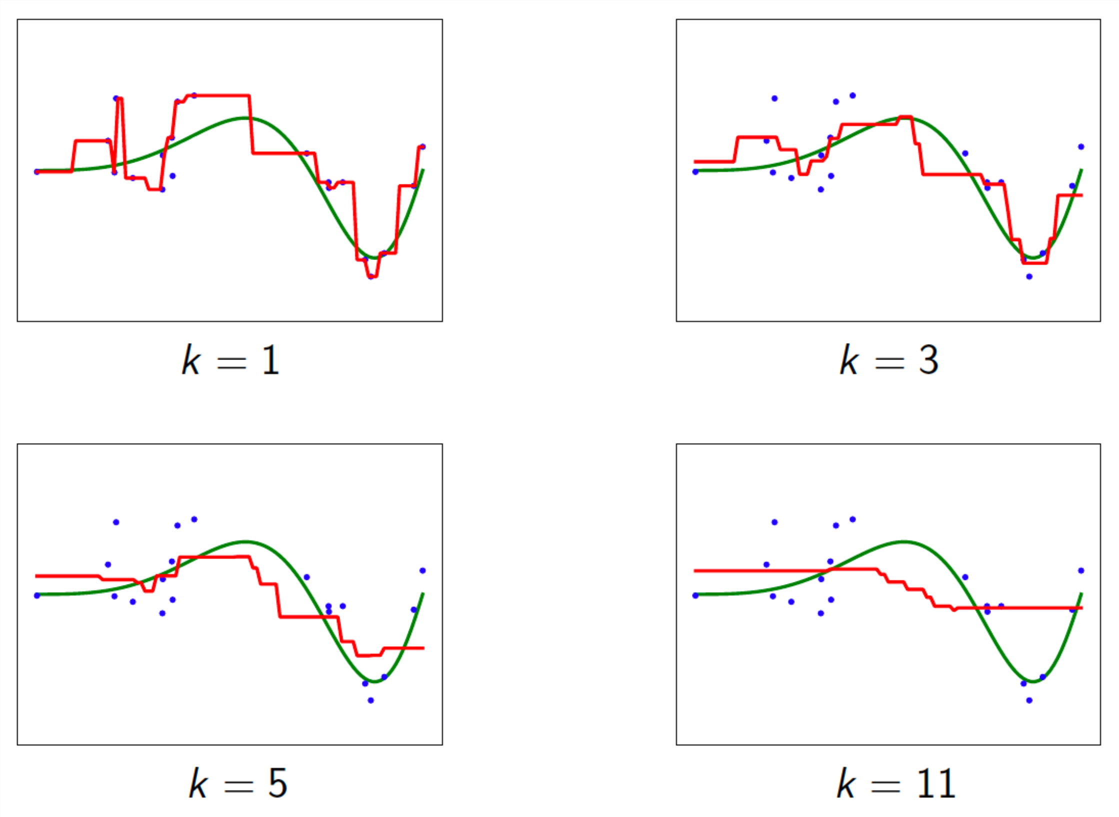
procedure kNN_REGRESSION($\mathcal{D}$, $\boldsymbol{x}$, k)
  # $\mathcal{D} = \{(\boldsymbol{x}_n, t_n)\}_{n=1}^N$ is training data
  # $\boldsymbol{x}$ is a test point, k is an integer
  sort $\mathcal{D}$ by increasing distance $d(\boldsymbol{x}_n, \boldsymbol{x})$
  $\mathbb{N}_k(\boldsymbol{x}) \leftarrow$ first k elements of sorted $\mathcal{D}$
  return $\displaystyle\frac{1}{k} \sum_{\boldsymbol{x}_n \in \mathbb{N}_k(\boldsymbol{x})} t_n$

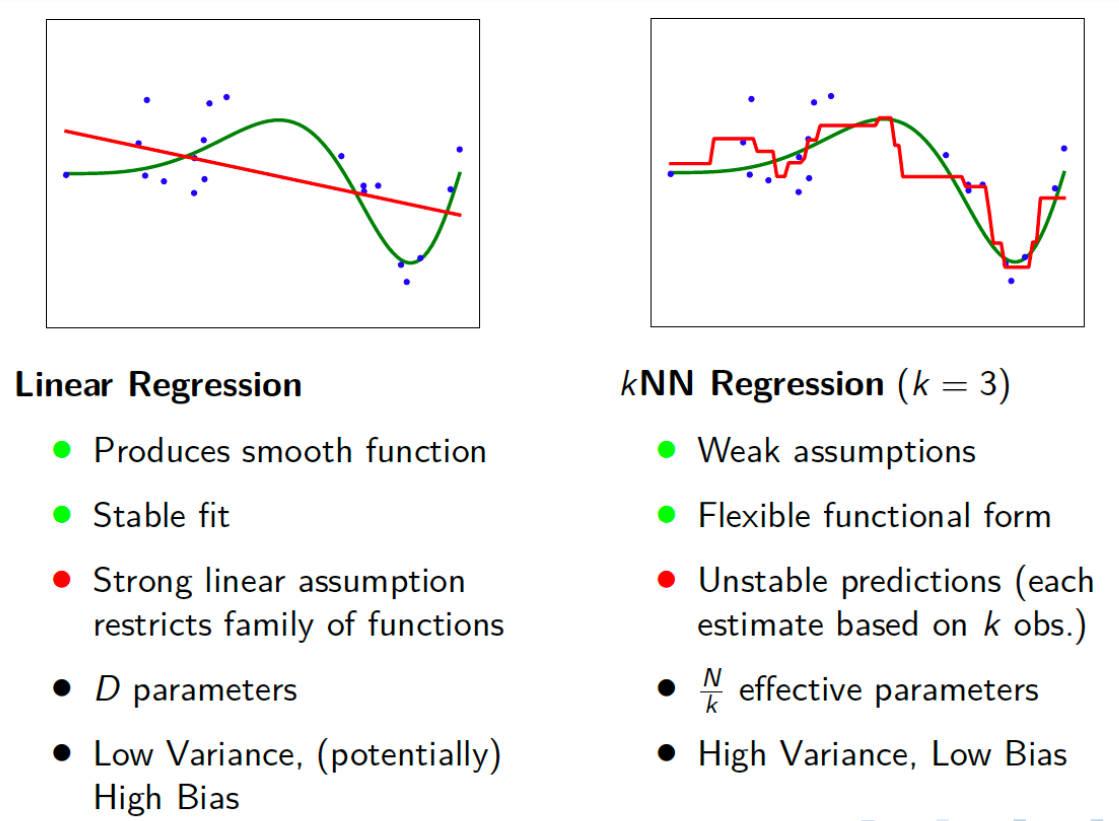
Insights

- No training phase
- Evaluation is expensdive, sort is $O(N \log N)$
- Seems like it has one parameter, k

- Actually has $\frac{N}{k}$ effective paramters

One-Dimensional kNN





Linear Models

Consider a ***simple linear regression*** with vector inputs:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{d=1}^{D} w_d x_d$$

with vector input data, $\mathbf{x} = (x_1, \ldots, x_D)$.

> Linear in both ***weights*** and ***input variables*** x_i

Extending that to consider:

$$y(\mathbf{x}, \mathbf{w}) = w_o + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where $\phi_j(\mathbf{x})$ are ***basis functions***. For instance, a monomial function: $\phi_j(\mathbf{x}) = \sum_i x_i^j$

> Also a ***linear model*** (linear in the weights, $\mathbf{w}$)

Extended linear model:

$$y(\mathbf{x}, \mathbf{w}) = w_o + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x})$$

where $\phi_0(\mathbf{x}) = 1$

Rewrite in vector form as:

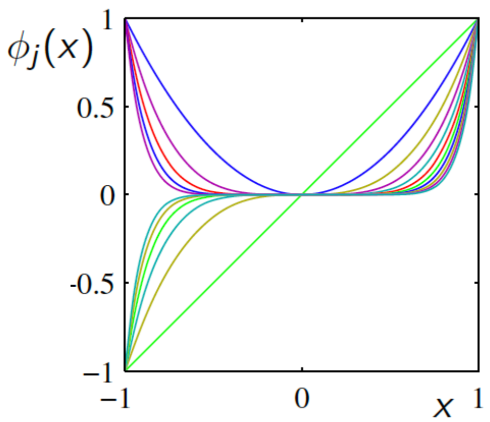
> ***Linear Model Prediction***
>
> $$y(\mathbf{x}, \mathbf{w}) = \phi(\mathbf{x})^T \mathbf{w}$$
>
> where $\phi(\mathbf{w})$ is our ***feature vector***, defined as:
>
> $$\phi(\mathbf{x}) = (\phi_o(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x}))^T$$

Example: Polynomial Basis Funtcion

We can choose our basis functions very flexibly:

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j\, \phi_j(\boldsymbol{x})$$

- In 1d: $\phi_j(x) = x^j$
- Generally: $\phi_j(\boldsymbol{x}) = \prod_d x_d^{j_d}$
- **global functions** – change in one region of input space affects all others



Example: Radial Basis Function

(Gaussian) Radial Basis Functions (RBF) are very common:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \, \phi_j(\mathbf{x})$$

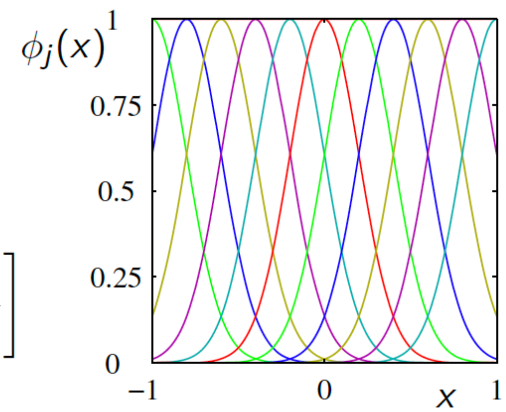
- For 1d input x:

  $\phi_j(x) = \exp\left(-(x-\mu_j)^2/2s^2\right)$

- Generally:

  $\phi_j(\mathbf{x}) = \exp\left[-\frac{(\mathbf{x}-\boldsymbol{\mu}_j)^T(\mathbf{x}-\boldsymbol{\mu}_j)}{2s^2}\right]$

- RBF effects are **local**.



Example: S-Shape Function

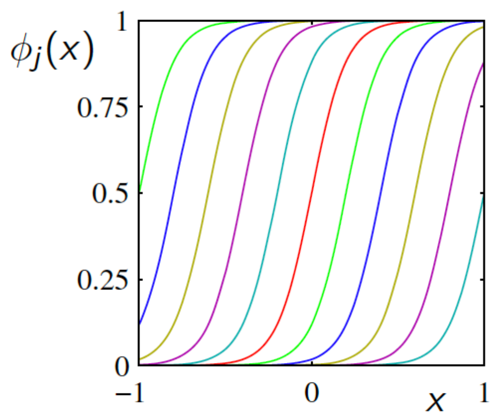
There are other choices of basis function for linear models:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \, \phi_j(\mathbf{x})$$

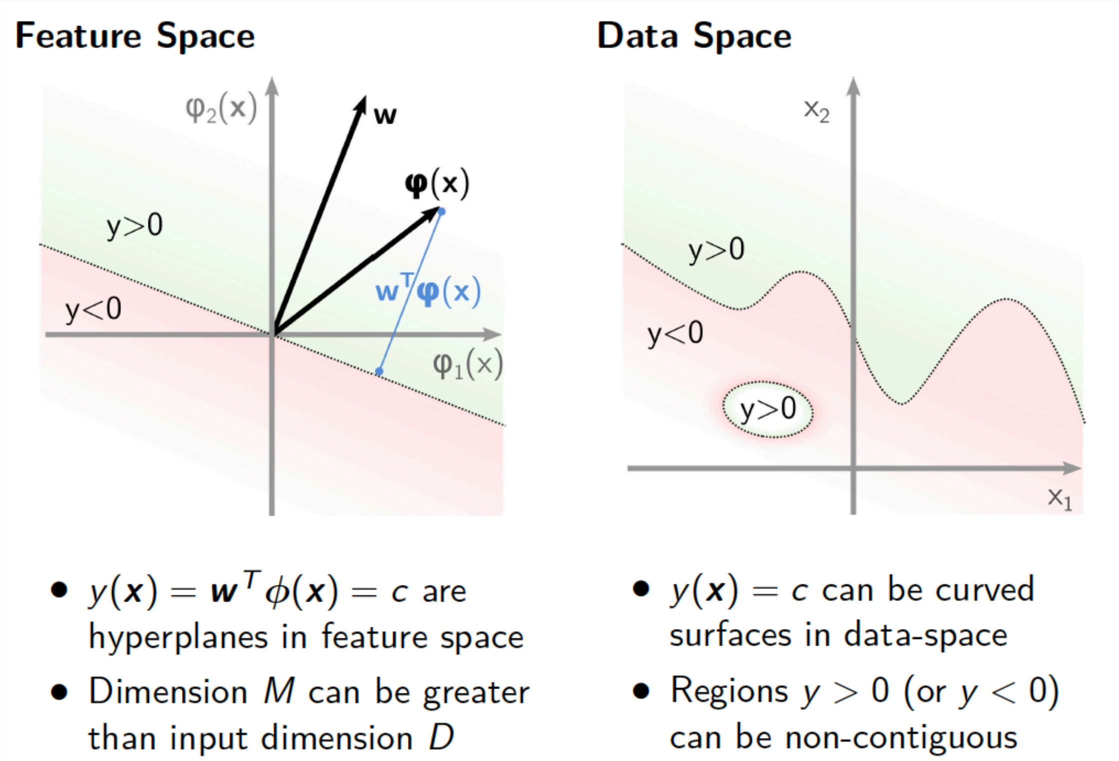
- The logistic function:

  $$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

  where $\sigma(a) = \dfrac{1}{1+\exp(-a)}$
- Similar functions, e.g. tanh
- Multidimensional?



Linear Models: Geometric Intuition



Fitting Linear Models

Assuming target t given by deterministic component plus ***Gaussian noise***:

$$t = y(\mathbf{x}; \mathbf{w}) + \epsilon$$

where $y(\mathbf{x}; \mathbf{w}) = \phi(\mathbf{x}_n)^T \mathbf{w}$ and $\epsilon \sim N(.|0, \beta^{-1})$

> The ***probability density*** for target t:
>
> $$p(t|\mathbf{x}, \mathbf{w}, \beta) = N(t|\phi(\mathbf{x}_n)^T \mathbf{w}, \beta^{-1})$$
>
> The ***conditional mean***:
>
> $$E[t|\mathbf{x}, \mathbf{w}, \beta] = \int tp(t|\mathbf{x}, \mathbf{w}, \beta)dt = y(\mathbf{x}, \mathbf{w})$$

Collect all inputs together into data matrix $\mathbf{X} = (x_1^T, \dots, x_N^T)^T$ with vector of corresponding targets $\mathbf{t} = (t_1, \dots, t_N)^T$ now likelihood is :

$$p(\mathbf{t}|\mathbf{X}, \beta) = \prod_{n=1}^{N} N(t_n|\phi(\mathbf{x}_n)^T \mathbf{w}, \beta^{-1})$$

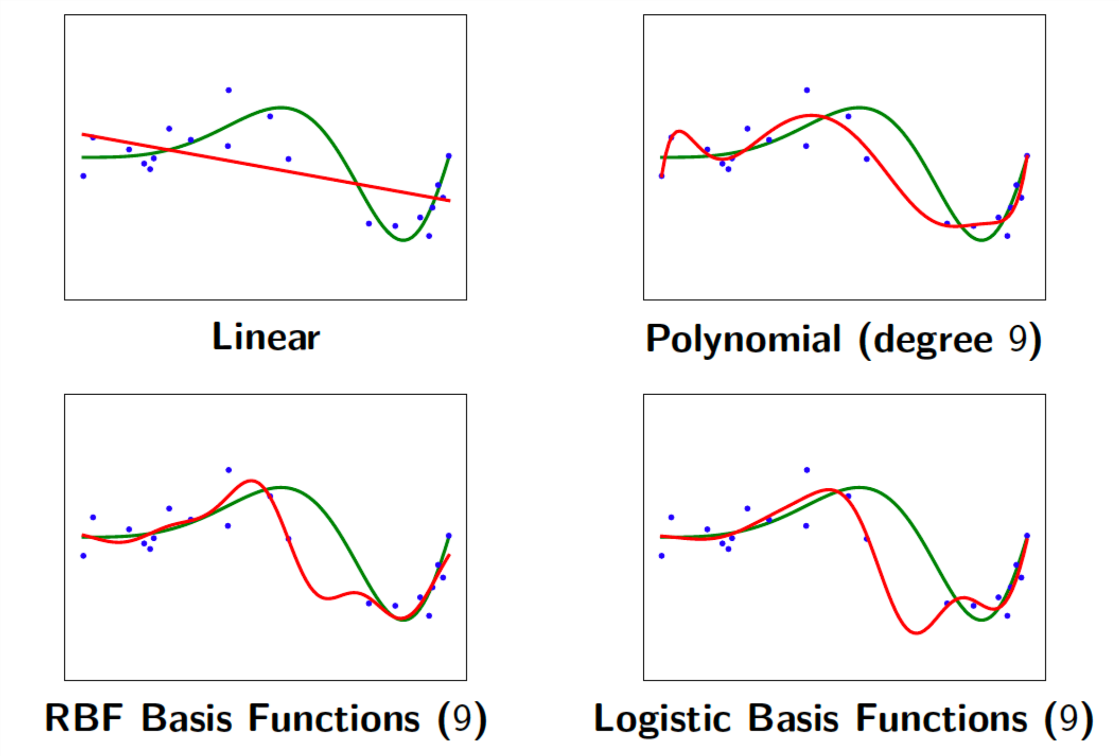
Taking log:

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{X}, \beta) &= \sum_{n=1}^{N} \ln N(t_n|\phi(\mathbf{x}_n)^T \mathbf{w}, \beta^{-1}) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w}) \end{aligned}$$

with ***sum-of-squares error***:

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

Fitting with Different Basis Functions



Finding the Maximum Likelihood

Since the function is ***quadratic*** in $\mathbf{w}$, there is a single maximum. To maximise the likelihood, differentiate and set to zero:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{X}, \beta) = \beta \nabla_{\mathbf{w}} E_D(\mathbf{w}) = 0$$

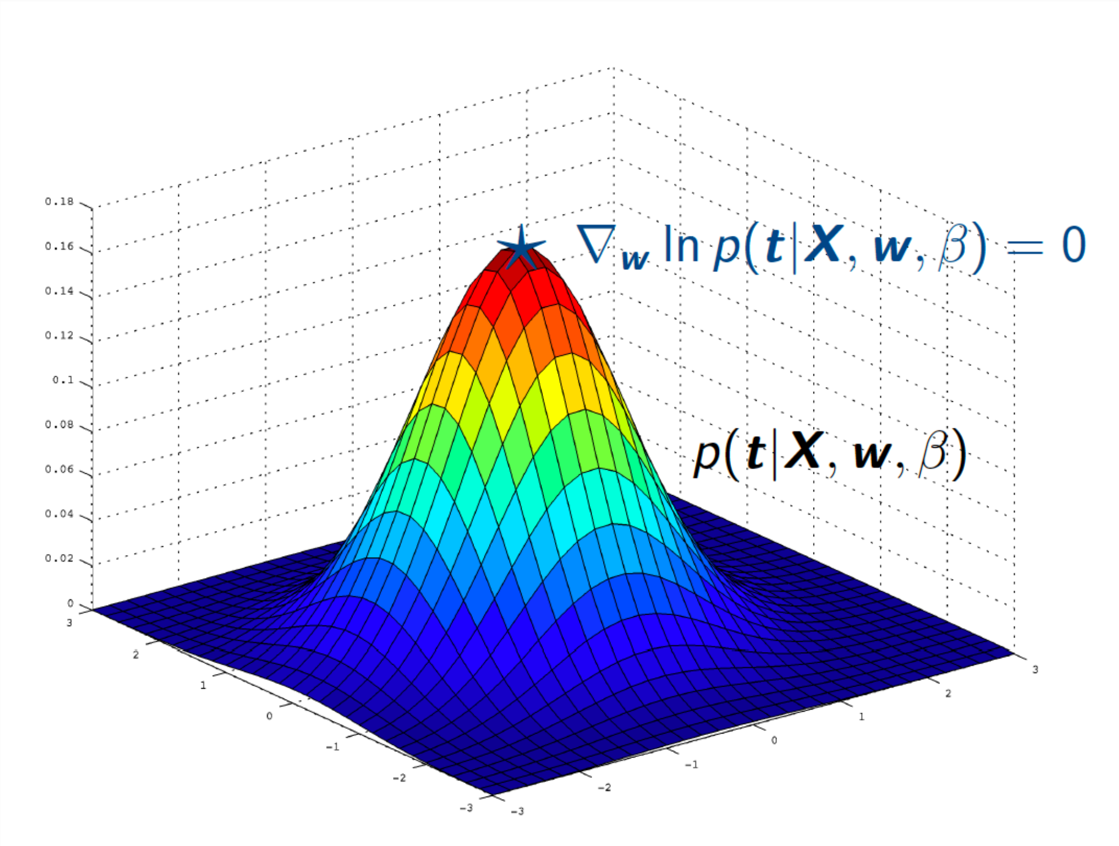
Rewrite ***error function*** in matrix form, differentiate and set to zero:

$$\begin{aligned}
E_D(\mathbf{w}) &= \frac{1}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w}) \\
\nabla_{\mathbf{w}} E_D(\mathbf{w}) &= -\boldsymbol{\Phi}^T(\mathbf{t} - \boldsymbol{\Phi}\mathbf{w}) = 0 \\
\Phi^T\Phi\mathbf{w} &= \Phi^T\mathbf{t} \\
\cancel{(\Phi^T\Phi)^{-1}\Phi^T\Phi}\ \mathbf{w} &= (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t} = \Phi^\dagger\mathbf{t}
\end{aligned}$$

where we have defined the design matrix as:

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \dots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \dots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \dots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

> ***ML Weights Linear Model:***
>
> $$\mathbf{w}_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t} = \Phi^\dagger\mathbf{t}$$



Regularised Least Squares

Regularisation by introducing an ***error term that penalises large weight values***:

$$\tilde{E}(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

As before:

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

Ridge Regression

For ***ridge regression***, weight penalty is $\lambda||\mathbf{w}||^2$, giving:

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2}||\mathbf{w}||^2$$

where λ is the ***regularisation coefficient***, controlling the relative importance of the two error terms. Total error function is now:

$$\begin{aligned}\tilde{E}(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}(t_n - \phi(\mathbf{x}_n)^T\mathbf{w})^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \\ &= \frac{1}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}\end{aligned}$$

Differentiate, and set to zero:

$$\begin{aligned}\nabla_{\mathbf{w}}\tilde{E}(\mathbf{w}) = -\mathbf{\Phi}^T(\mathbf{t} - \mathbf{\Phi}\mathbf{w}) + \lambda\mathbf{w} &= 0 \\ -\Phi^T\mathbf{t} + \Phi^T\Phi\mathbf{w} + \lambda\mathbf{w} &= 0 \\ (\Phi^T\Phi + \lambda I)\mathbf{w} &= \Phi^T\mathbf{t} \\ \mathbf{w} &= (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t}\end{aligned}$$

The error function for ***ridge regression*** (also know as ***weight decay***) is quadratic which means it has a closed form solution too:

> **Regularised Weights Linear Model***:
>
> $$\mathbf{w}^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t}$$

Lasso

Other regularisation terms are also possible. For instance, ***sum-of-absolute-values***:

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{M}|w_j|$$

- This approach is know as ***lasso***
- If λ is sufficiently large, can lead to a ***sparse model*** : where most weight coefficients w_j are exactly zero.
- Sparse models can be more ***robust*** (resistant to over-fitting)

> No general closed form solution for $\mathbf{w}$