Supervised learning in non-linear settings

Statistical Computing and Empirical Methods
Unit EMATM0061, Data Science MSc

Rihuan Ke
rihuan.ke@bristol.ac.uk

Teaching Block 1, 2024

What we will cover today

We will investigate a small selection of different approaches for non-linear regression.

We will begin with an approach for applying linear methods to non-linear problems with random non-linear feature mappings.

We will then consider the neural network approach where the feature mapping is learnt from the data.

We will also talk about the k nearest neighbour method - a flexible distance-based approach.

A simple non-linear regression problem

Learning a regression model

Assume that we have a pair of random variables (X, Y) with unknown distribution $\mathbf{P}$.

- $X \in \mathcal{X}$ denotes a feature vector and $Y \in \mathbb{R}$ denotes a continuous variable.

Aim: learn a regression model ϕ that takes X as input and has an output that is close to Y.

We can do this by minimising the mean squared error of regression models

$$\mathcal{R}(\phi) := \mathbb{E}\big[(\phi(X) - Y)^2\big]$$

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

However, we don't know $\mathbf{P}$, so we can not compute directly the optimal function η.

Learning a regression model

Assume that we have a pair of random variables (X, Y) with unknown distribution $\mathbf{P}$.

- $X \in \mathcal{X}$ denotes a feature vector and $Y \in \mathbb{R}$ denotes a continuous variable.

If we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$ i.i.d.

We can learn from the data a regression model $\hat{\phi}$ that minimises the training error

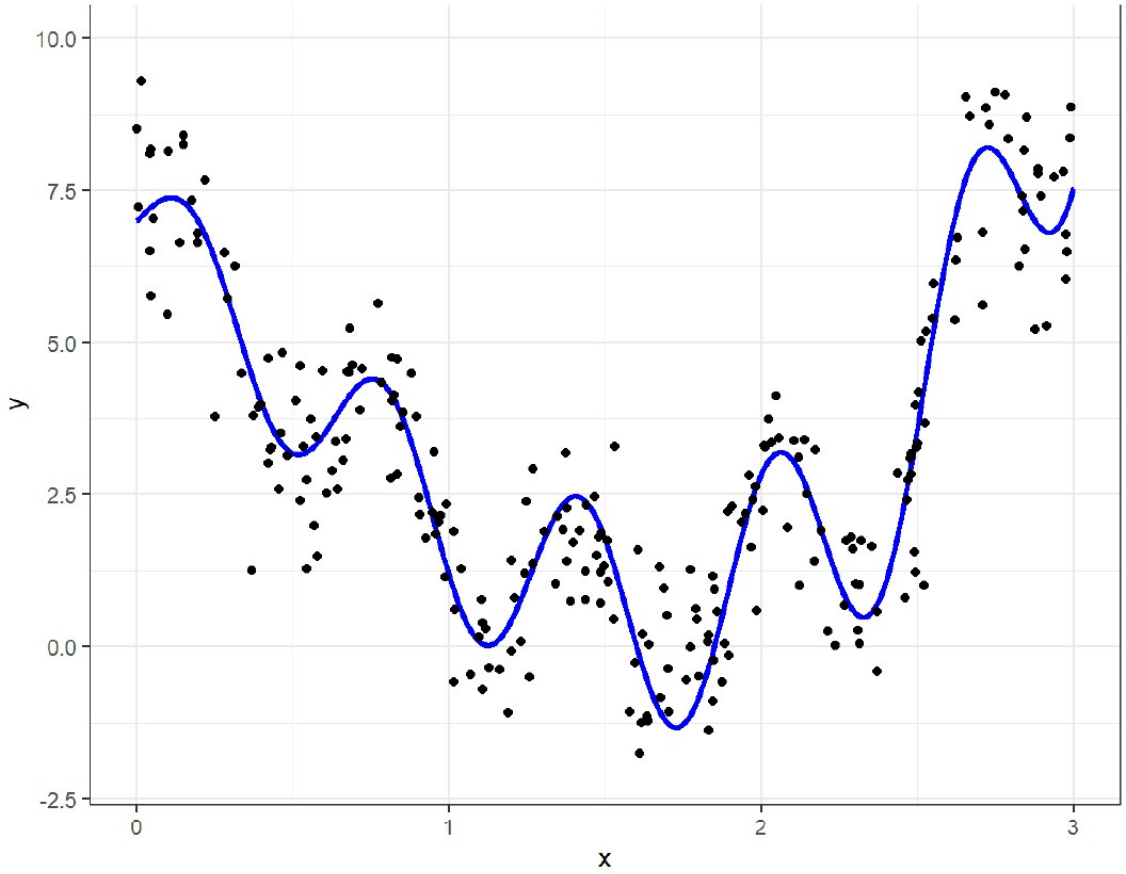
$$\hat{\mathcal{R}}_{\text{MSE}}(\phi) := \frac{1}{n} \sum_{i=1}^{n} (\phi(X_i) - Y_i)^2$$

We don't just want to do well on training data & achieve a low training error

Doing well on unseen data corresponds to having $\hat{\phi}(X) \approx \eta(X)$ for typical $(X, Y) \sim \mathbf{P}$.

A simple regression problem

Let's consider the following regression problem.



(X_i, Y_i)

$(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

We want to find a function $\phi(x)$ that "looks similar" to $\eta(x)$.

Linear regression methods

Previously we considered linear regression models of the form $\phi_{w,w^0} : \mathbb{R}^d \to \mathbb{R}$ defined by

$$\phi_{w,w^0}(x) = xw^T + w^0$$

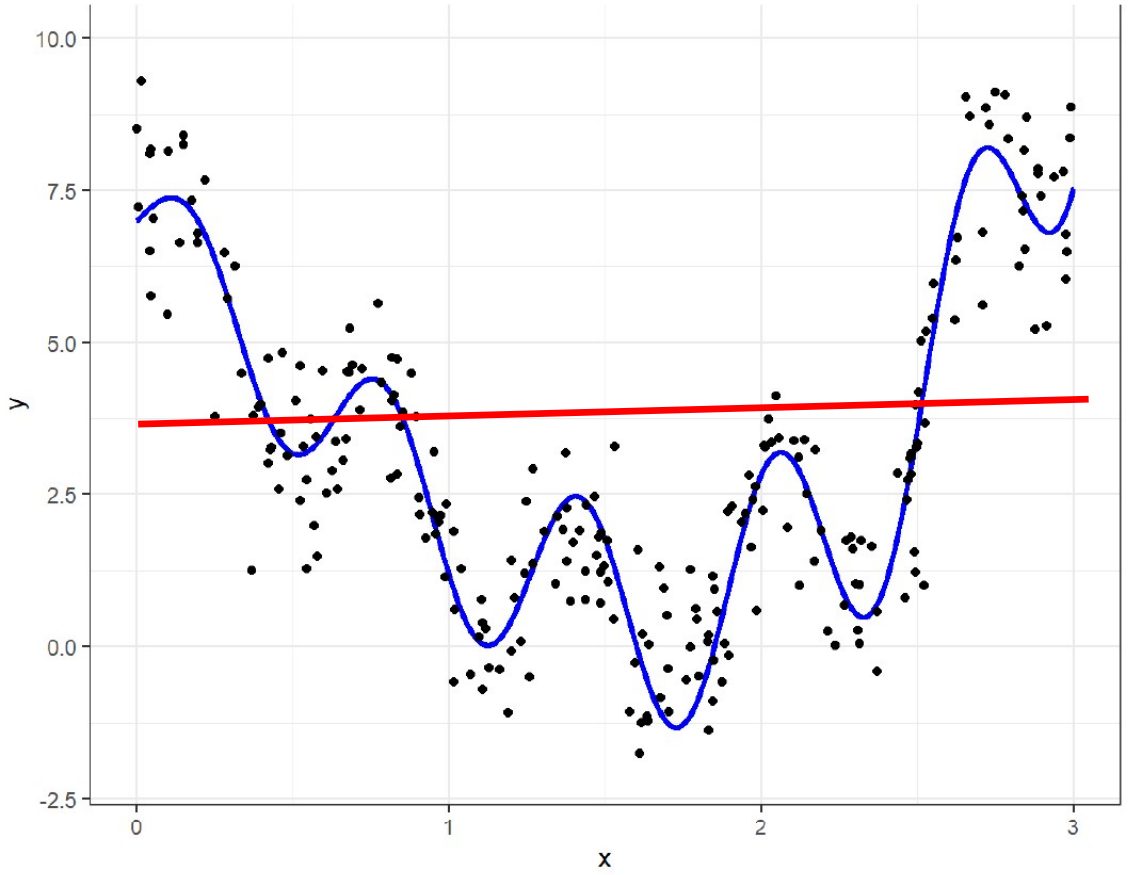
where $w = (w^1, \cdots, w^d) \in \mathbb{R}^d$ is a weight vector and $w^0 \in \mathbb{R}$ is a bias term.

Example of learning algorithms for this includes

1) Ordinary Least Squares.
2) Ridge regression.
3) Lasso regression.
4) Elastic net regression.

A simple regression problem

Linear methods yield poor results for many problems (especially when the problem itself is non-linear)



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}_{\hat{w},\hat{w}_0}(x))$

We need non-linear regression models.

Three approaches to non-linear problems

Three approaches to non-linear problems:

Approach 1: First apply a stochastic transformation of the feature space. Then apply a linear technique in the transformed feature space.

Approach 2: Jointly learn a transformation of the feature space and a regression model.

Approach 3: Apply a non-parametric technique based on the proximity of test feature vectors to training examples.

The first two approaches are parametric methods, and they are both based on non-linear feature mappings.

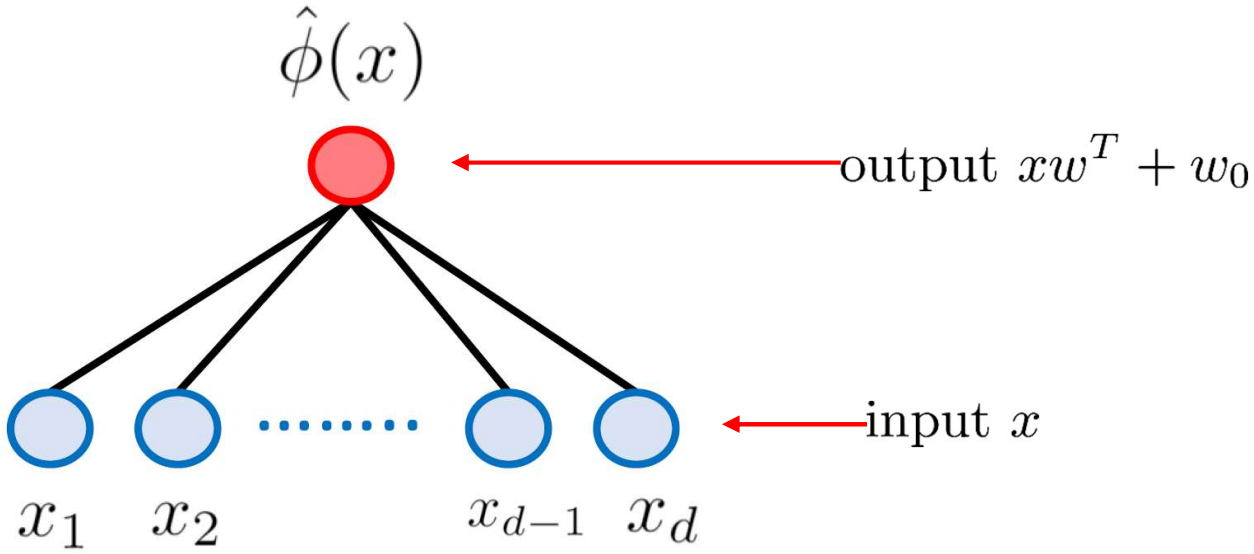
Non-linear feature mappings

Linear regression models

We considered linear regression models $\phi_{w,w_0} : \mathbb{R}^d \to \mathbb{R}$ of the form defined by

$$\phi_{w,w_0}(x) = w_1 x_1 + \cdots + w_d x_d + w_0 = x w^T + w_0$$

Linear mapping:

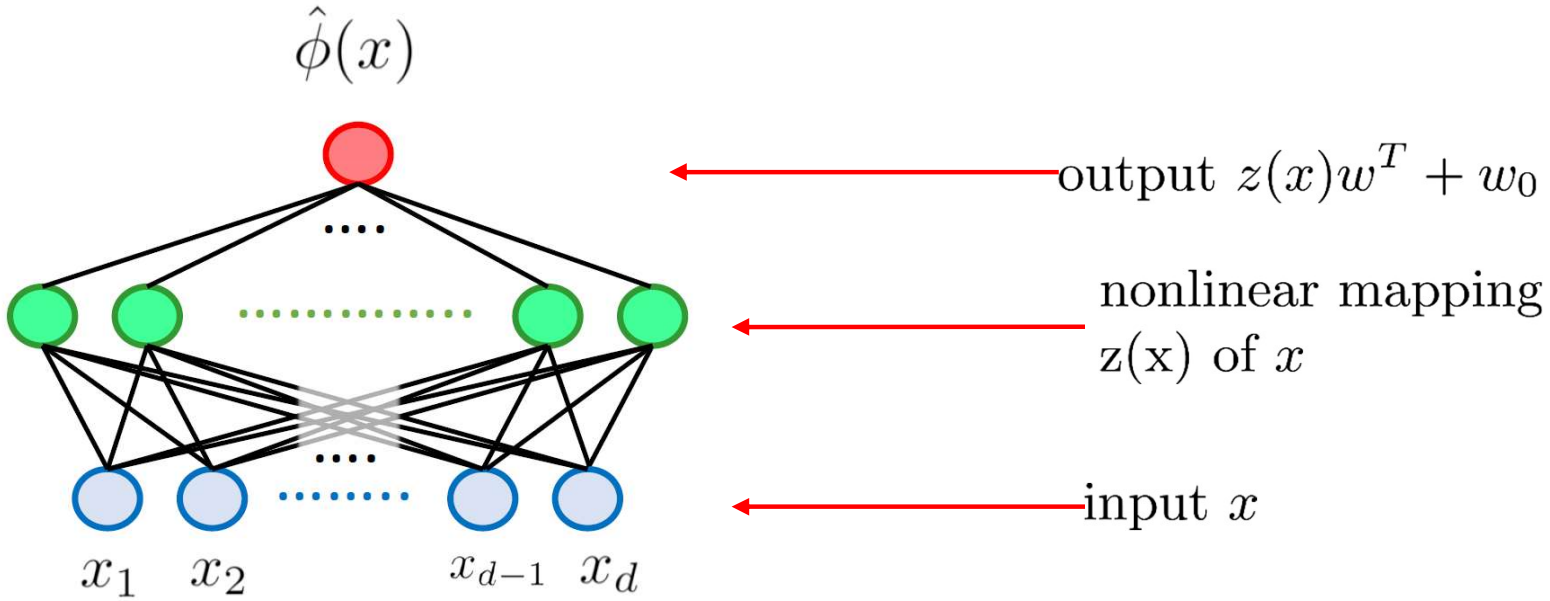


Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) := z(x)w^T + w_0$$

Here $z(x)$ is one kind of non-linear mapping of the features.



$z(x)$ can be constructed with a linear mapping and an activation function (nonlinear). See the next slide.

Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) := z(x)w^T + w_0$$

Here $z(x)$ is one kind of non-linear mapping of the features.

First, take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d} \qquad b = (b_1, \cdots, b_q) \in \mathbb{R}^q$$

Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) := z(x)w^T + w_0$$

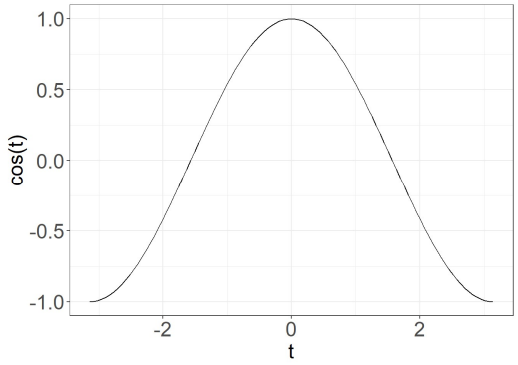
Here $z(x)$ is one kind of non-linear mapping of the features.

First, take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

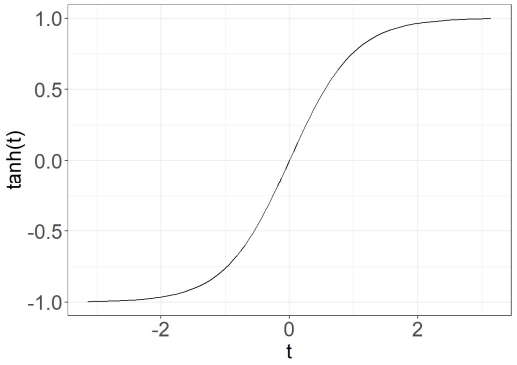
Second, choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

Examples:

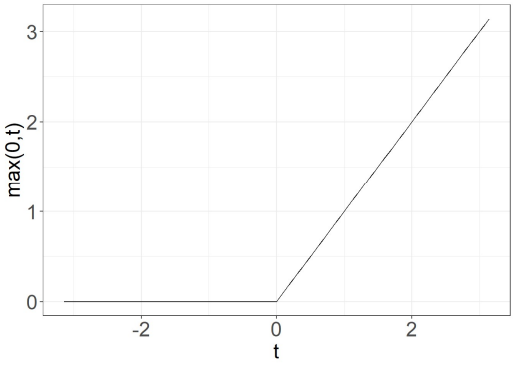
1). $\psi(t) = \cos(t)$



2). $\psi(t) = \tanh(t)$



3). $\psi(t) = \max(0, t)$



Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) := z(x)w^T + w_0$$

Here $z(x)$ is one kind of non-linear mapping of the features.

First, take a weight matrix $A \in \mathbb{R}^{q \times d}$ and a bias vector $b = (b_1, \cdots, b_q) \in \mathbb{R}^q$.

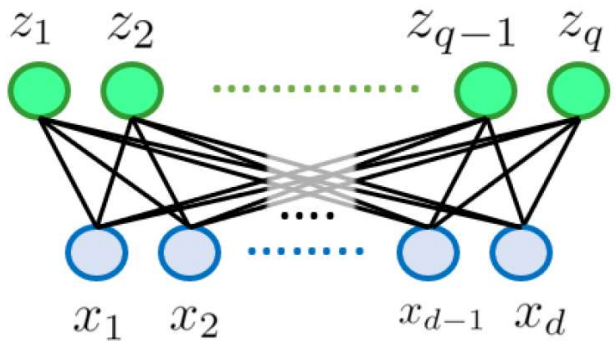
Second, choose a non-linear activation function $\psi : \mathbb{R} \to \mathbb{R}$.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d}$$

$$b = (b_1, \cdots, b_q) \in \mathbb{R}^q$$

$z(x) := (z_1(x), z_2(x), \cdots, z_q(x))$ is a vector in $\mathbb{R}^q$ defined by:

$$\begin{aligned} z_1(x) &= \psi(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1d}x_d + b_1) \\ z_2(x) &= \psi(a_{21}x_1 + a_{22}x_2 + \cdots + a_{2d}x_d + b_2) \\ &\vdots \\ z_q(x) &= \psi(a_{q1}x_1 + a_{q2}x_2 + \cdots + a_{qd}x_d + b_q) \end{aligned}$$



We can rewrite this as $z(x) = \psi(xA^T + b)$.

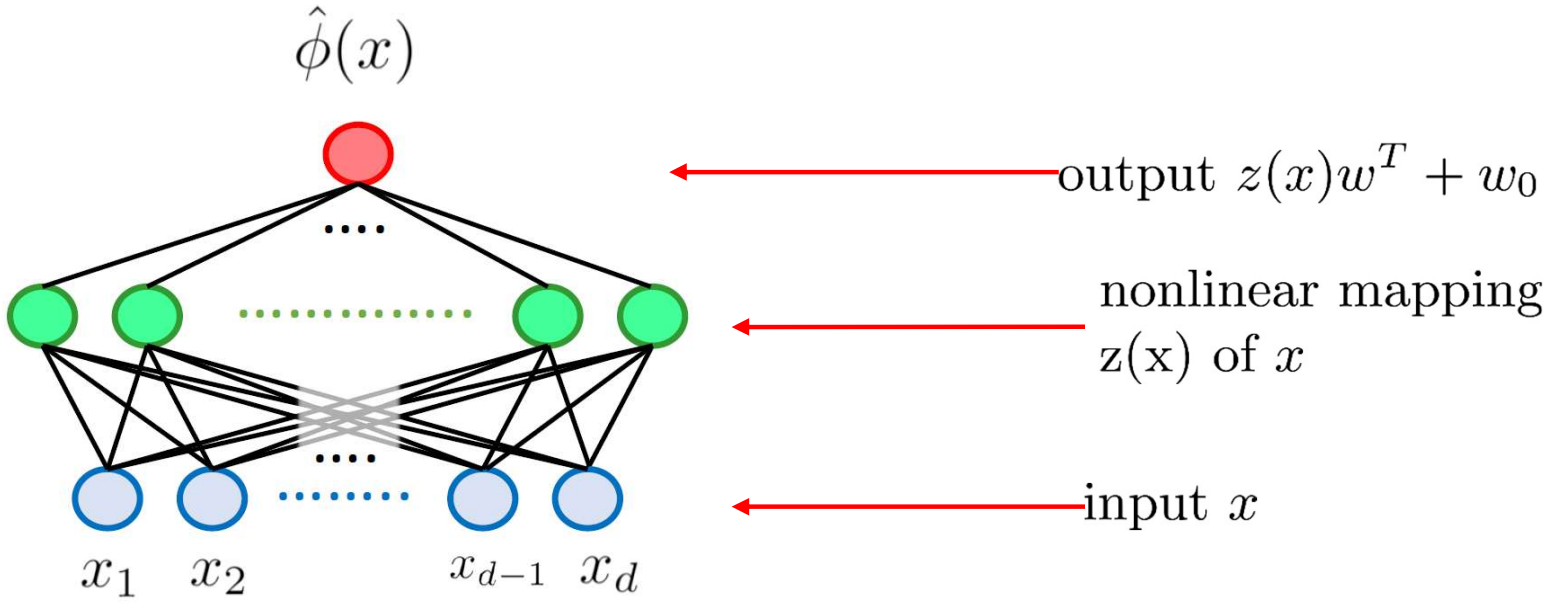
Note: $xA^T + b \in \mathbb{R}^q$ is a linear transform of x.

Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) := z(x)w^T + w_0$$

Here $z(x)$ is one kind of non-linear mapping of the features.



$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0$ is nonlinear.

Non-linear feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

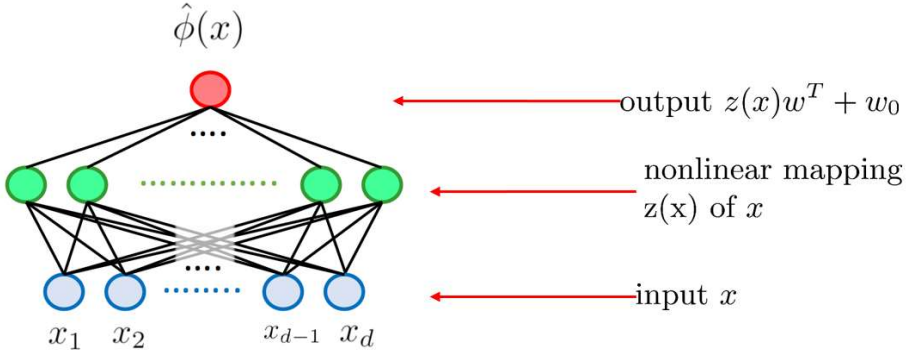
$$\hat{\phi}(x) := z(x)w^T + w_0$$

Here $z(x) = \psi(xA^T + b)$ is a non-linear mapping of the features.

Examples of $\hat{\phi}$

1). Neural networks (A and b are parameters that are adjustable during learning).

2). Random feature mapping methods (A and b are randomly generated and fixed).



Random feature mappings

Random feature mappings

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0.$$

Approach 1: First apply a stochastic transformation of the feature space. Then apply a linear technique in the transformed feature space.

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and bias $b \in \mathbb{R}^q$.

2. Fix feature mappings $z(x) = \psi(xA^T + b)$

3. Use the data $((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ to learn a linear mapping

$$\hat{\phi}(x) = z(x)w^T + w_0$$

Ridge regression with random feature mappings

The regression model is

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0 \text{ is nonlinear.}$$

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and bias $b \in \mathbb{R}^q$.

We choose $\sigma > 0$, and

$$\text{randomly sample} \quad A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d} \quad \text{with} \quad a_{rs} \sim \mathcal{N}(0, \sigma^2)$$

$$\text{randomly sample} \quad b = (b_1, \cdots, b_q) \in \mathbb{R}^q \quad \text{with} \quad b_r \sim \text{Unif}([0, 2\pi)) \text{ i.i.d.}$$

Ridge regression with random feature mappings

The regression model is

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0 \text{ is nonlinear.}$$

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and bias $b \in \mathbb{R}^q$.

   randomly sample A and b with $\quad a_{rs} \sim \mathcal{N}(0, \sigma^2)$ and $b_r \sim \text{Unif}([0, 2\pi))$

2. Fix feature mappings $z(x) = \psi(xA^T + b)$

   define $\psi : \mathbb{R} \to \mathbb{R}$ by $\psi(t) := \cos(t)$.

   So for each X_i, we can define $Z_i := z(X_i)$.

3. Use the data $((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ to learn a linear mapping

   $$\hat{\phi}(x) = z(x)w^T + w_0$$

   In this step, $z(x)$ is fixed, so we only need to learn w and w_0.

   Given the transformed data $((Z_1, Y_1), \cdots, (Z_n, Y_n))$, this is a linear regression problem! We can apply the ridge regression method to find w and w_0.

Ridge regression with random feature mappings

The regression model is

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0 \text{ is nonlinear.}$$

1. Randomly choose weights $A \in \mathbb{R}^{q \times d}$ and bias $b \in \mathbb{R}^q$.

2. Fix feature mappings $z(x) = \psi(xA^T + b)$ So $Z_i := z(X_i)$.

3. Use the data $((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ to learn a linear mapping

   Applying $z(\cdot)$ to X_i, and define a new dataset $\mathcal{D}_z := ((Z_1, Y_1), \cdots, (Z_n, Y_n))$

   Applying the ridge regression to $\mathcal{D}_z$ to find w and w_0. $\hat{\phi}(x) = z(x)w^T + w_0$

This method has two hyper-parameters:

(1) The noise parameter σ for the random weights in A

(2) The norm penalty λ in the ridge regression

These can be optimised by performance on the validation set.

Ridge regression with random feature mappings in R

Suppose we have a train, validation and test split of our data set.

Let's extract feature vectors from labels as follows (for training, validation, and test dataset respectively):

```
train_x<-train_data%>%select(-y)%>%as.matrix()
train_y<-train_data%>%pull(y)
```

```
val_x<-val_data%>%select(-y)%>%as.matrix()
val_y<-val_data%>%pull(y)
```

```
test_x<-test_data%>%select(-y)%>%as.matrix()
test_y<-test_data%>%pull(y)
```

Ridge regression with random feature mappings in R

Next we generate our random feature mapping $z(x) = \psi(xA^T + b)$ where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d} \quad \text{with} \quad a_{rs} \sim \mathcal{N}(0, \sigma^2)$$

$$b = (b_1, \cdots, b_q) \in \mathbb{R}^q \quad \text{with} \quad b_r \sim \text{Unif}([0, 2\pi)) \text{ i.i.d.}$$

```
num_random_features<-500
sigma=0.01
```

q

A
b

```
d<-train_x%>%ncol()
rand_weights_matrix<-matrix(rnorm(num_random_features*d,sd=sigma),ncol=d)
rand_bias<-runif(num_random_features,max=(2*pi))
```

Ridge regression with random feature mappings in R

Next we generate our random feature mapping $z(x) = \psi(xA^T + b)$ where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{q1} & a_{q2} & \cdots & a_{qd} \end{pmatrix} \in \mathbb{R}^{q \times d} \quad \text{with} \quad a_{rs} \sim \mathcal{N}(0, \sigma^2)$$

$$b = (b_1, \cdots, b_q) \in \mathbb{R}^q \quad \text{with} \quad b_r \sim \text{Unif}([0, 2\pi)) \text{ i.i.d.}$$

$z(x) = \psi(xA^T + b)$

```
random_feature_transform<-function(x){
  n_x<-x%>%nrow()
  z<-cos(x%*%t(rand_weights_matrix)+
           matrix(rep(rand_bias,n_x),nrow=n_x,byrow=TRUE))
  return(z)
}
```

Ridge regression with random feature mappings in R

We want to learn $\hat{\phi}(x) = z(x)w^T + w_0$

Suppose that we have training data $\mathcal{D} := ((X_1, Y_1), \cdots, (X_n, Y_n))$

Applying $z(\cdot)$ to X_i, and define a new dataset $\mathcal{D}_z := ((Z_1, Y_1), \cdots, (Z_n, Y_n))$

```
train_z<-random_feature_transform(train_x)
```

$Z_1, \cdots, Z_n$

Applying the ridge regression to $\mathcal{D}_z$ to find w and w_0.

```
library(glmnet)
lambda<-10
```

```
nonlinear_ridge_model<-glmnet(x=train_z,y=train_y,lambda=lambda,alpha=0)
```

$\hat{\phi} = zw^T + w_0$ $Z_1, \cdots, Z_n$ $Y_1, \cdots, Y_n$

Ridge regression with random feature mappings in R

We want to learn $\hat{\phi}(x) = z(x)w^T + w_0$

Suppose that we have training data $\mathcal{D} := ((X_1, Y_1), \cdots, (X_n, Y_n))$

Applying $z(\cdot)$ to X_i, and define a new dataset $\mathcal{D}_z := ((Z_1, Y_1), \cdots, (Z_n, Y_n))$

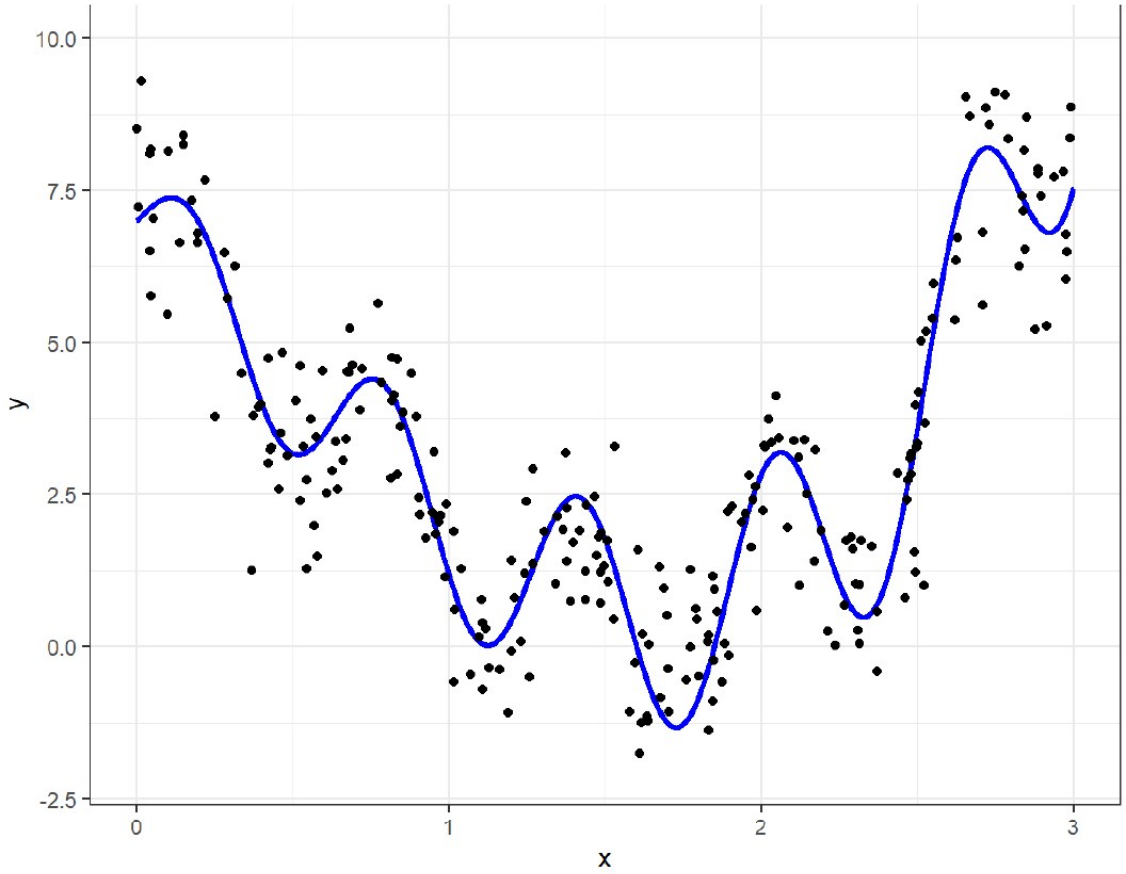
Applying the ridge regression to $\mathcal{D}_z$ to find w and w_0.

We can then apply our model to validation (or test data) and compute the mean squared error.

```
val_z<-random_feature_transform(val_x)
val_predictions<-predict(nonlinear_ridge_model,newx=val_z)
val_msq_error<-mean((val_predictions-val_y)^2)
```

Ridge regression with random feature mappings in R

Returning to our example regression problem.



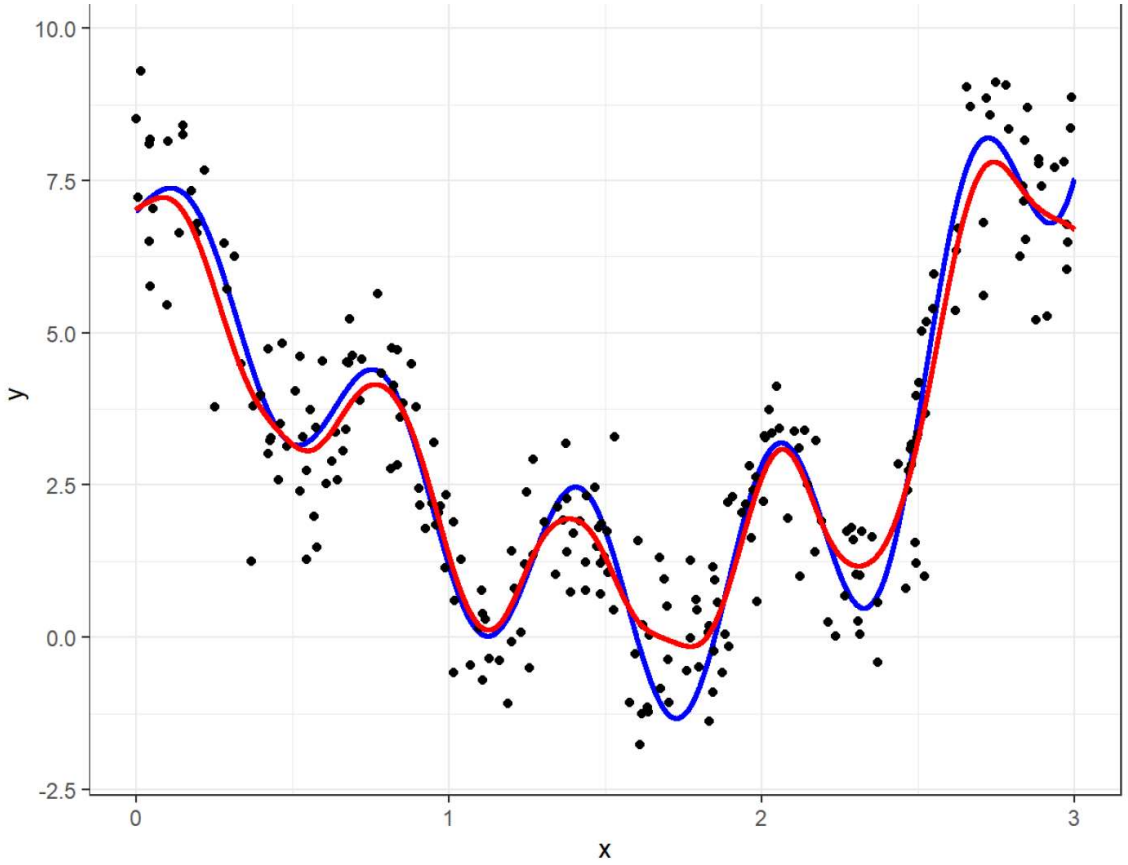
(X_i, Y_i)

$(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Ridge regression with random feature mappings in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

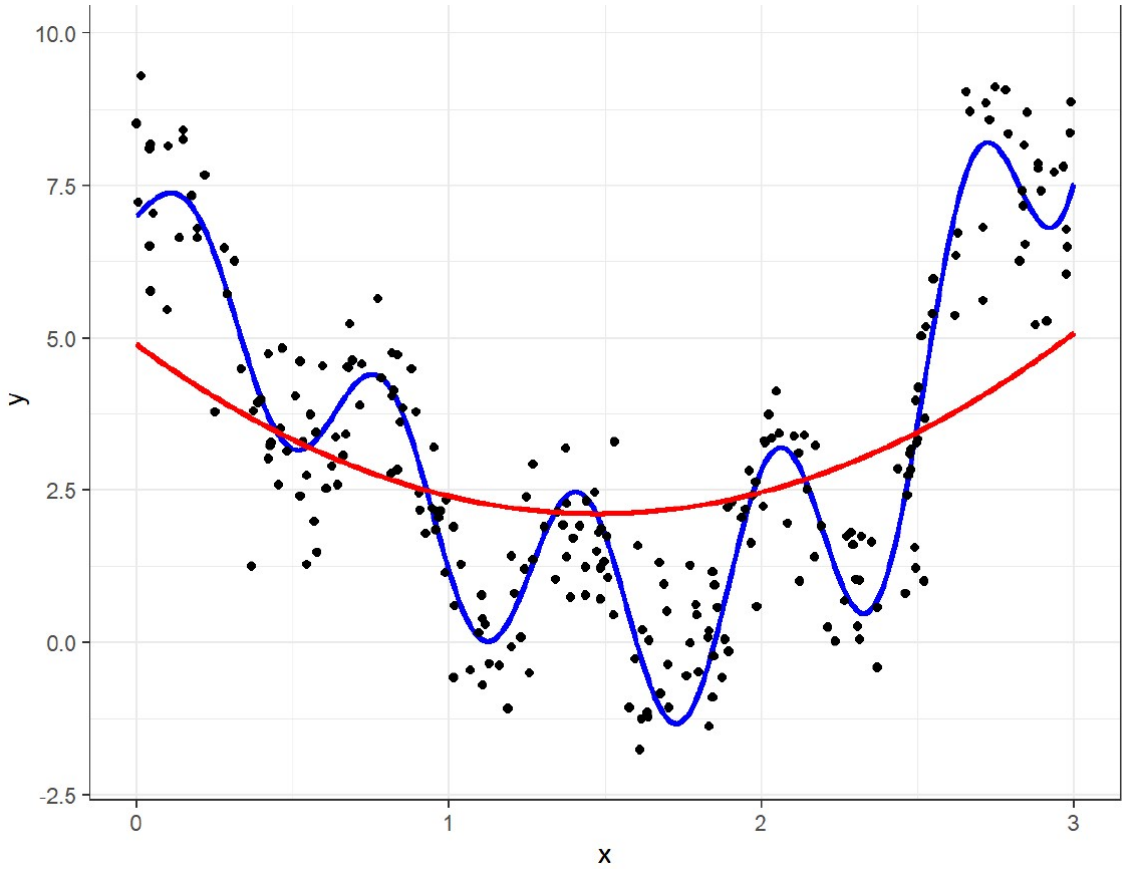
$(x, \hat{\phi}(x))$

with $\sigma = 10$ and $\lambda = 10$.

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Ridge regression with random feature mappings in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

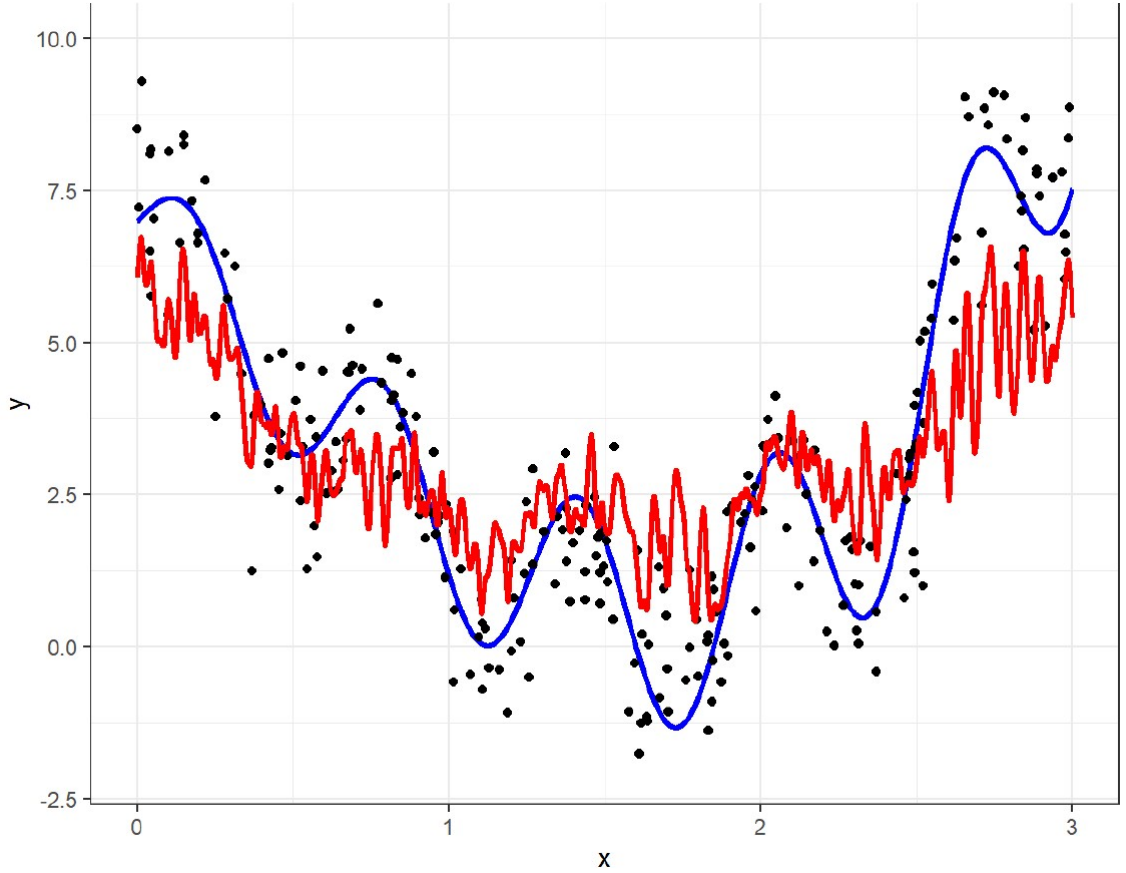
$(x, \hat{\phi}(x))$

with $\sigma = 0.01$ and $\lambda = 10$.

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Ridge regression with random feature mappings in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

with $\sigma = 100$ and $\lambda = 10$.

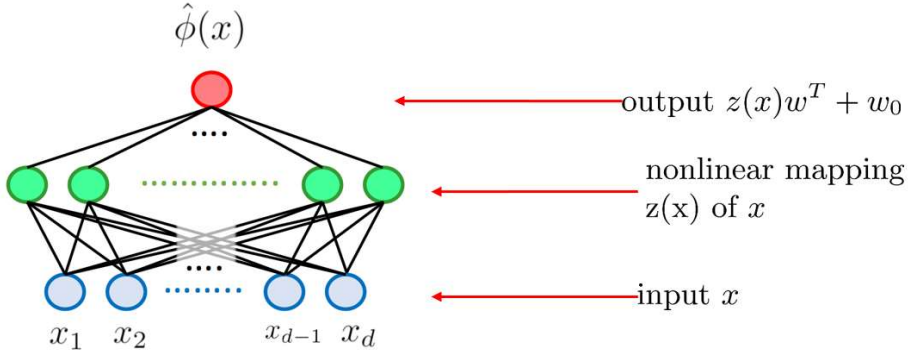
Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Nonlinear Regression based on Neural networks

Nonlinear regression based on neural networks

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0.$$



Approach 2: Jointly learn a transformation of the feature space and a regression model.

$\hat{\phi}$ represents a typical neural network

The matrix A and the bias b are both adjustable parameters of the neural network

We need to decide the parameters $\theta := (A, b, w, w_0)$ from the data.

Nonlinear regression based on neural networks

We can create non-linear regression models $\hat{\phi} : \mathbb{R}^d \to \mathbb{R}$ by combining linear mappings with simple nonlinear mappings.

$$\hat{\phi}(x) = z(x)w^T + w_0 = \psi(xA^T + b)w^T + w_0.$$

$\hat{\phi}$ represents a typical neural network

We need to decide the parameters $\theta := (A, b, w, w_0)$ from the data.

Our goal is to find a mapping $\phi : \mathcal{X} \to \mathbb{R}$ with a small test error

$$\mathcal{R}(\phi) := \mathbb{E}\big[(\phi(X) - Y)^2\big]$$

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

We can learn a regression model $\hat{\phi}$ by minimising the training error

$$\hat{\mathcal{R}}_{\text{MSE}}(\phi) := \frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - Y_i)^2$$

over all neural networks of the form $\phi(x) = \psi(xA^T + b)w^T + w_0$

Training neural networks

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

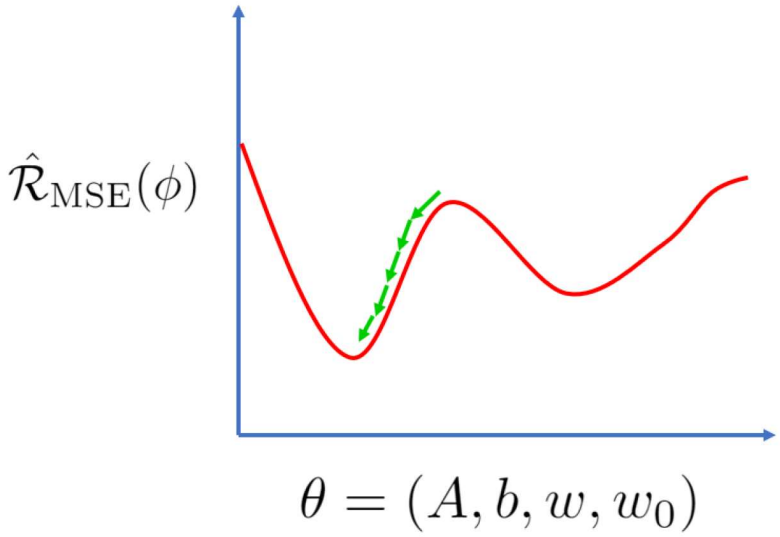
We can learn a regression model $\hat{\phi}$ by minimising the training error

$$\hat{\mathcal{R}}_{\text{MSE}}(\phi) := \frac{1}{n} \sum_{i=1}^{n} (\phi(X_i) - Y_i)^2$$

over all neural networks of the form $\phi(x) = \psi(xA^T + b)w^T + w_0$

We need find parameter $\theta := (A, b, w, w_0)$ that optimise $\hat{R}_{\text{MSE}}$

This can be done iteratively with gradient descent (numerical optimisation).



1. Initialize the parameters $\theta = \theta_0$ (e.g., with random numbers).

2. For $t = 0, \cdots, T-1$, update θ by

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial}{\partial \theta} \left\{ \hat{R}_{\text{MSE}}(\theta) \right\} \Big|_{\theta_t}.$$

Here α is a parameter controlling the step size.

Neural networks in R

We want to learn a regression model $\hat{\phi}$ by minimising the training error

$$\hat{\mathcal{R}}_{\text{MSE}}(\phi) := \frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - Y_i)^2$$

over all neural networks of the form $\phi(x) = \psi(xA^T + b)w^T + w_0$

We can create a simple neural network in R with the "nnet"

```
library(nnet) # load library
```

Set a random seed (for reproducibility) and specify a maximum number of iterations.

```
set.seed(123) # set random seed
```

```
max_iterations<-10000 # maximum number of iterations
```

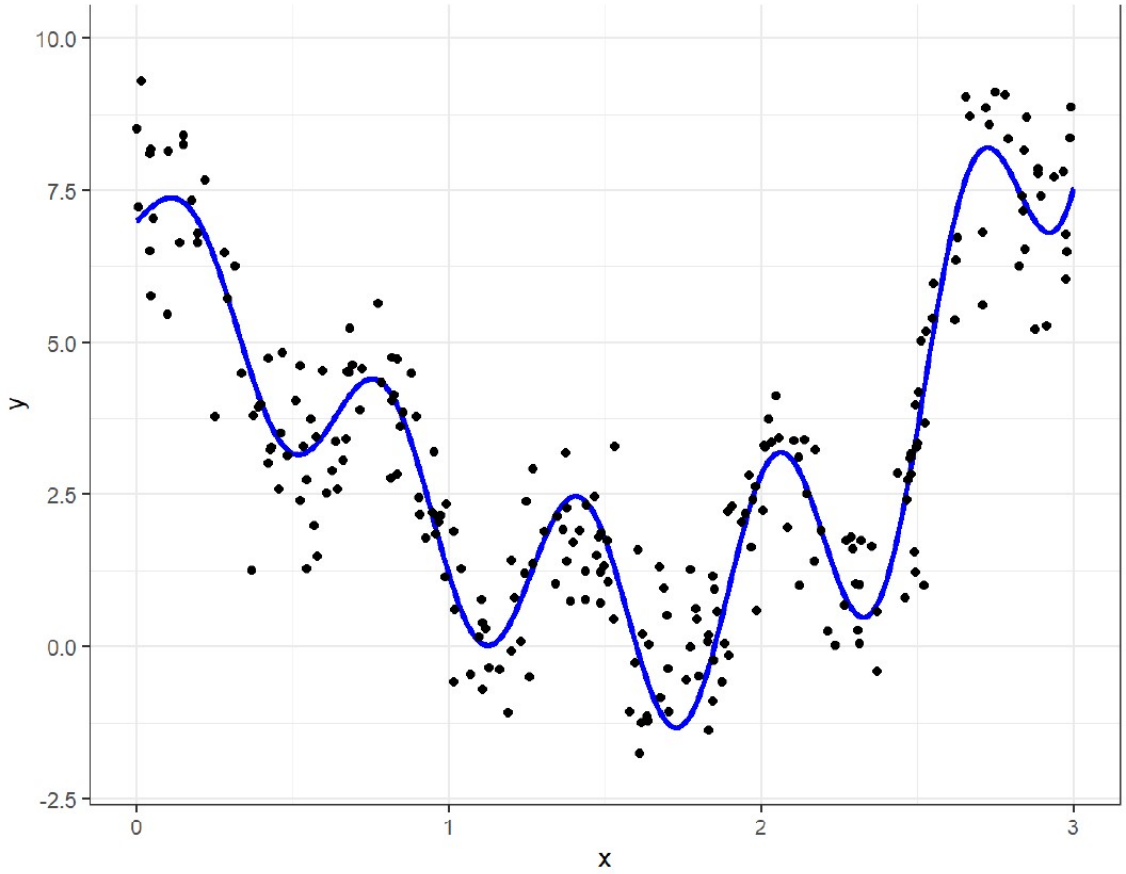
Train the neural network $Y_1, \cdots, Y_n$ q (the number of rows of A).

```
nnet_model<-nnet(train_x, train_y,size=500,
                 MaxNWts=2000,linout=TRUE,maxit=max_iterations) # train nnet
```

$X_1, \cdots, X_n$

Regression using neural networks in R

Returning to our example regression problem.



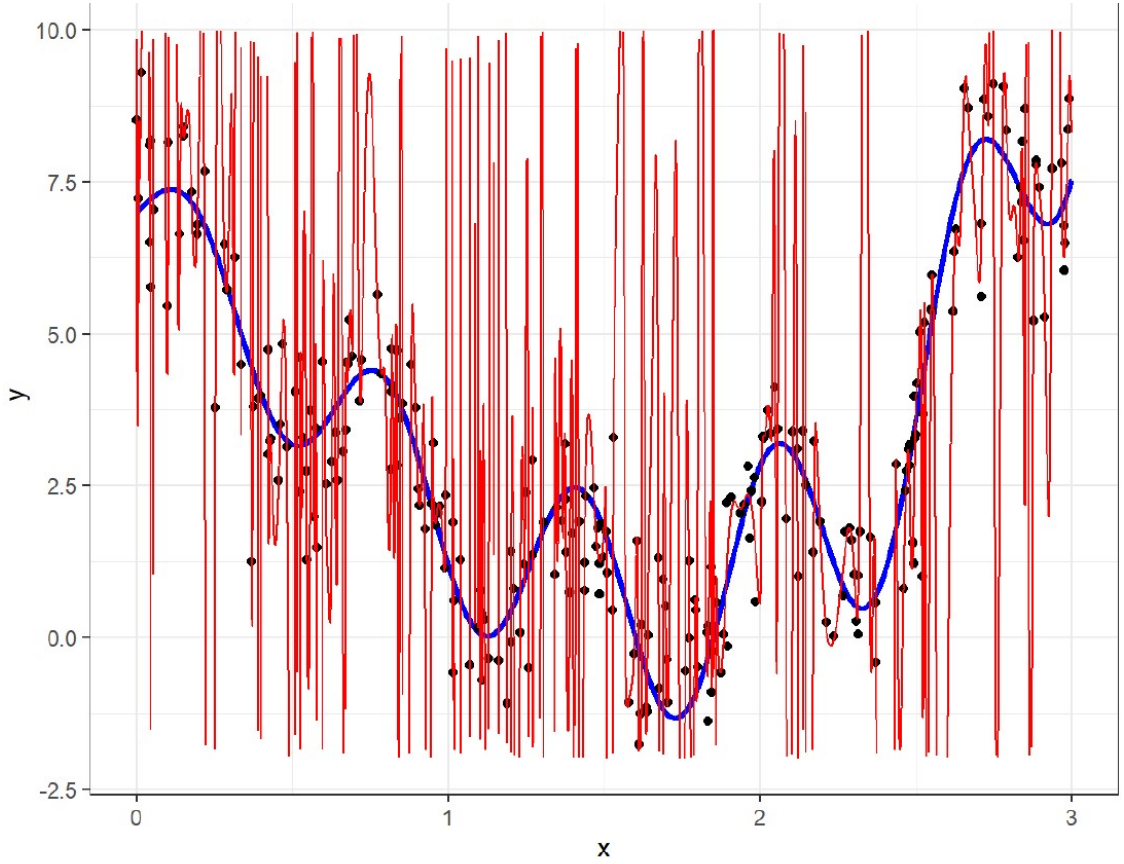
(X_i, Y_i)

$(x, \eta(x))$

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Regression using neural networks in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

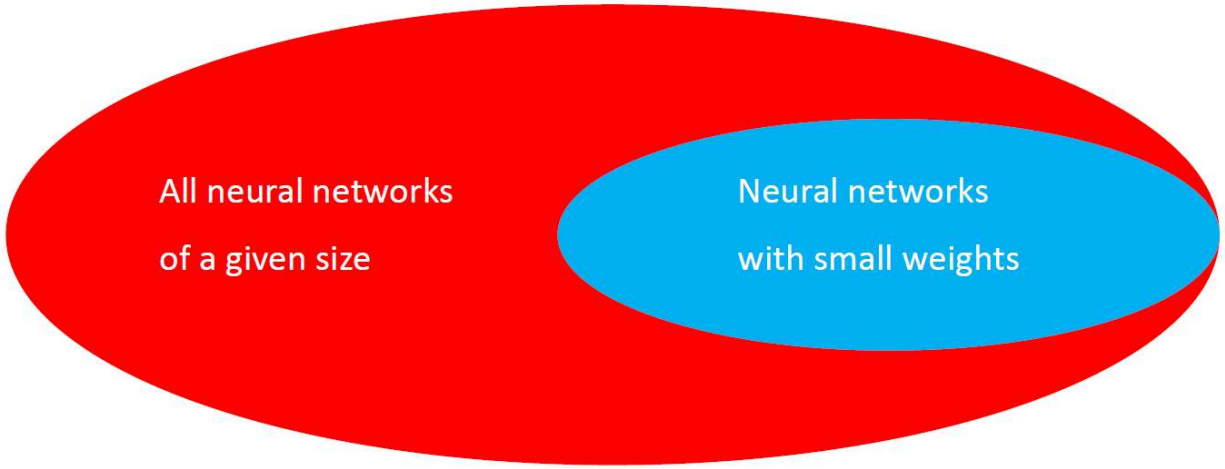
$(x, \hat{\phi}(x))$

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Neural networks with weight decay

The source of the instability is a search over a large space of all neural networks of a given size.

Idea: To reduce the instability of our estimate we limit our search space to neural networks with "small" weights (regularisation)



We focus on the space of neural networks with weights of low norm.

Neural networks with weight decay

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

Idea: To reduce the instability of our estimate we limit our search space to neural networks with "small" weights (regularisation)

We can learn a regression model $\hat{\phi}$ by minimising the regularised loss

$$\hat{\mathcal{R}}^{\lambda}_{\mathrm{MSE}}(\phi) := \underbrace{\frac{1}{n}\sum_{i=1}^{n}(\phi(X_i) - Y_i)^2}_{\text{data fitting term}} + \underbrace{\lambda\left(\|A\|_F^2 + \|w\|_2^2\right)}_{\text{regularisation term}}$$

over all neural networks of the form $\phi(x) = \psi(xA^T + b)w^T + w_0$

Here $\|A\|_F^2 = \sum_{r=1}^{q}\sum_{s=1}^{d}(a_{rs})^2$ and $\|w\|^2 = \sum_{r=1}^{q}(w_r)^2$.

Minimising the regularised loss $\hat{\mathcal{R}}^{\lambda}$ is known as l_2 regularisation or weight decay.

Neural networks with weight decay

$$\hat{\mathcal{R}}^{\lambda}_{\mathrm{MSE}}(\phi) := \underbrace{\frac{1}{n}\sum_{i=1}^{n}(\phi(X_i)-Y_i)^2}_{\text{data fitting term}} + \underbrace{\lambda\left(\|A\|_F^2+\|w\|_2^2\right)}_{\text{regularisation term}}$$

Let's set the weight decay and maximum iterations.

```
max_iterations<-10000 # maximum number of iterations
weight_decay<-0.01 # set weight decay
```

parameter λ.

Train the neural network to minimise the regularised loss function.

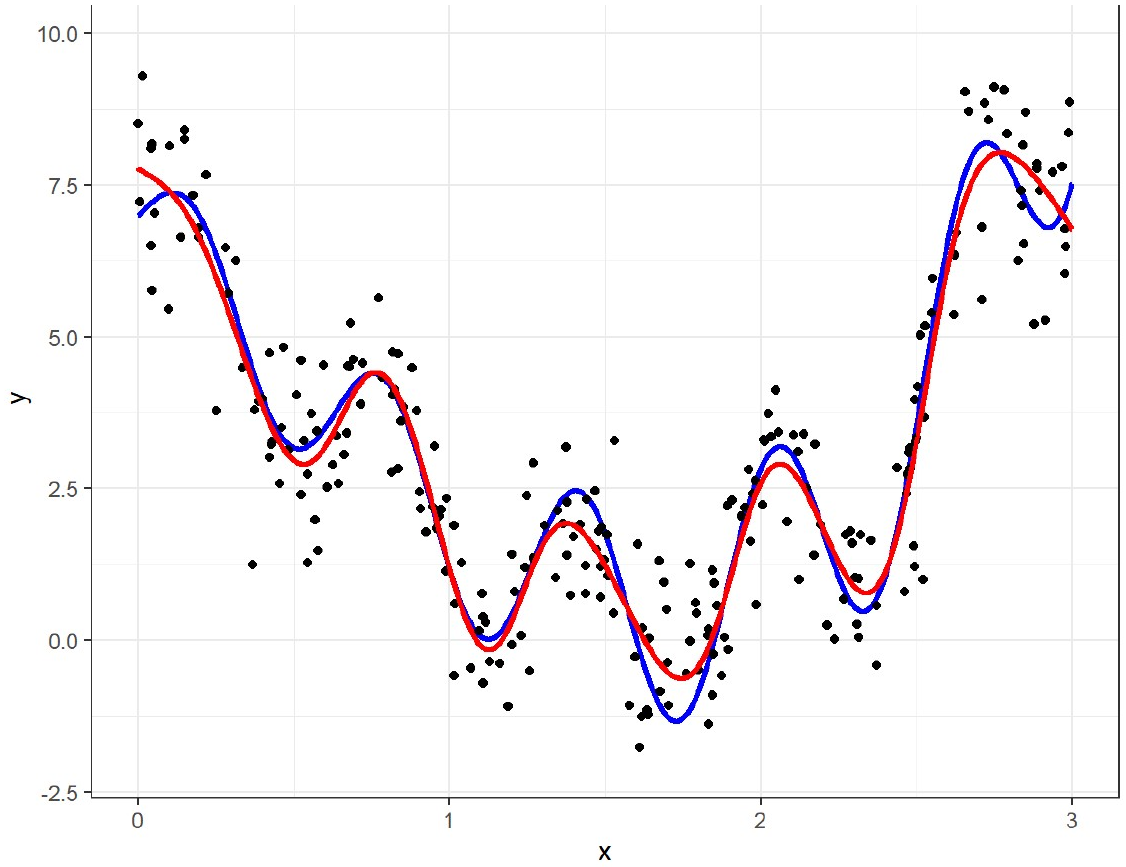
```
set.seed(123) # set random seed
nnet_model<-nnet(train_x, train_y,size=500,decay=weight_decay,
                 MaxNWts=2000,linout=TRUE,maxit=max_iterations) # train nnet
```

Use our trained model to make predictions and estimate the validation error.

```
val_predictions<-predict(nnet_model,newdata=val_x) # validation predictions
val_msq_error<-mean((val_predictions-val_y)^2)  # validation error
```

Regression using neural networks in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

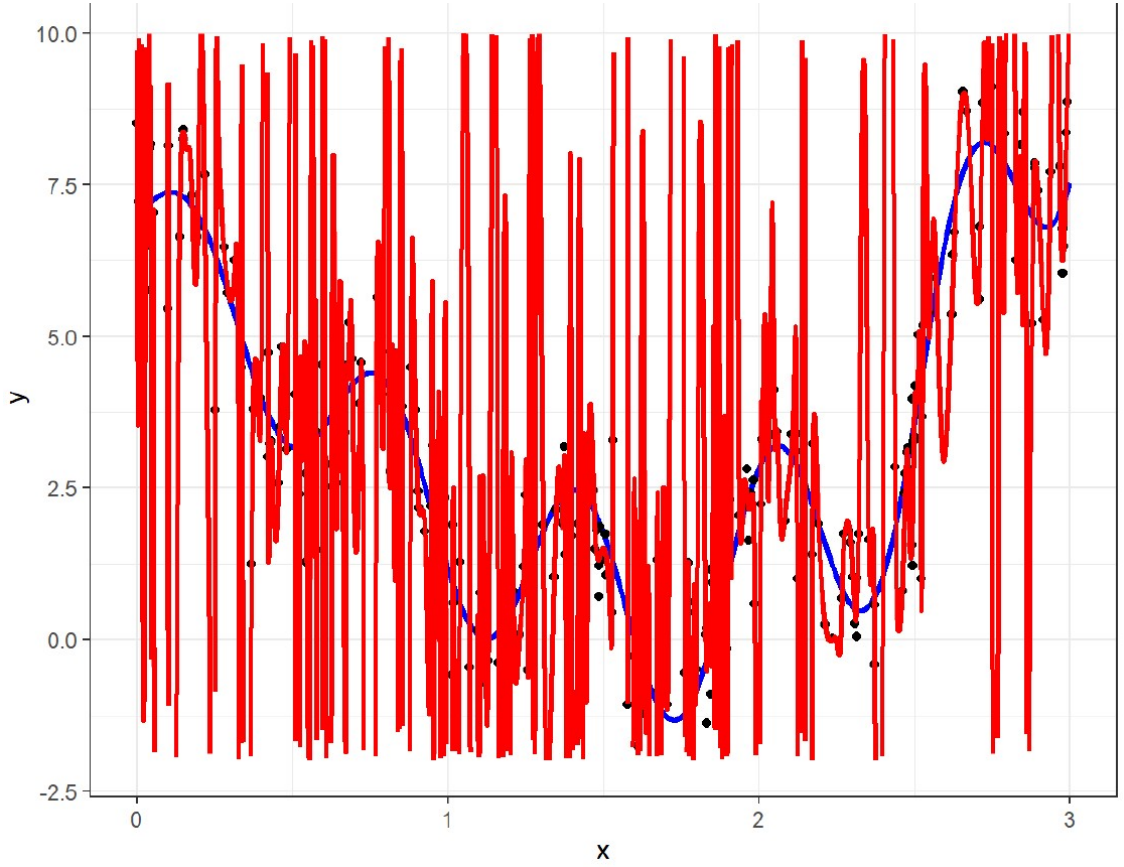
$(x, \hat{\phi}(x))$

weight decay $\lambda = 0.01$.

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Regression using neural networks in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

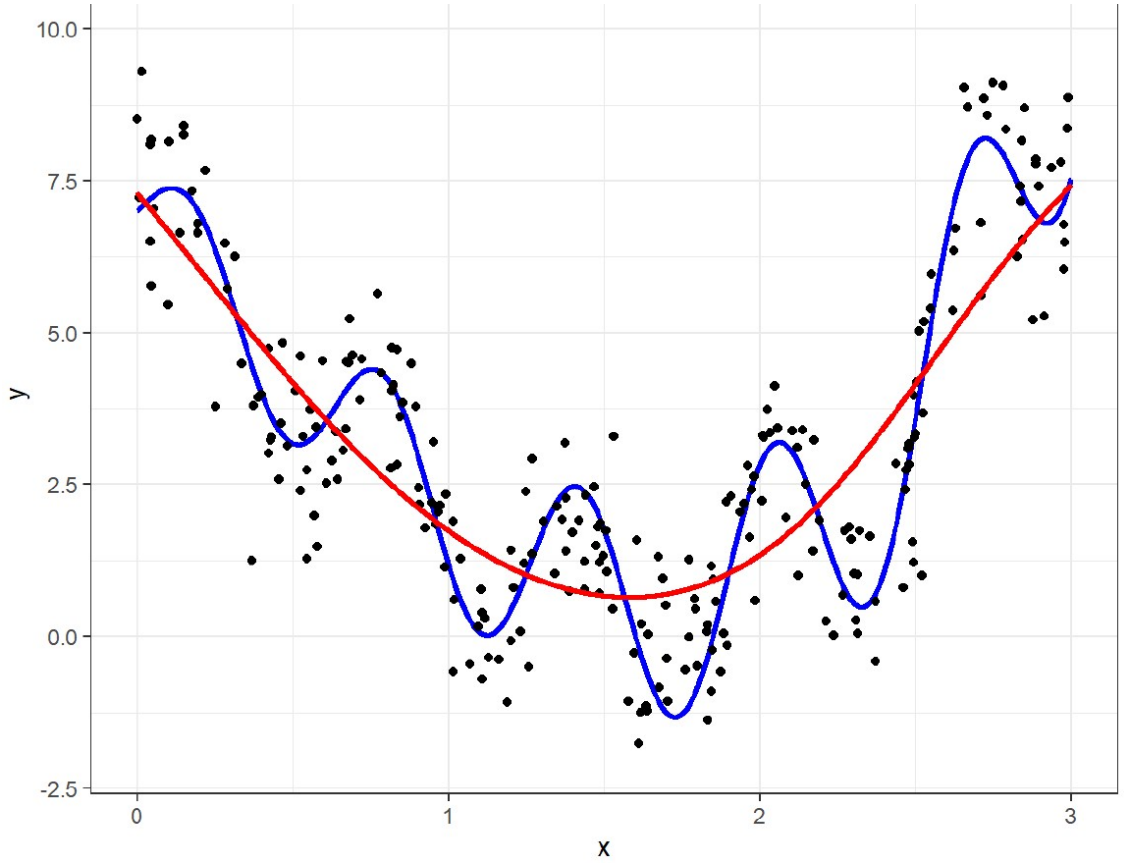
$(x, \hat{\phi}(x))$

weight decay $\lambda = 10^{-8}$.

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

Regression using neural networks in R

Returning to our example regression problem.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

weight decay $\lambda = 1$.

Here $\eta(x) = \mathbb{E}(Y \mid X = x)$ is the optimal predictive function.

A non-parametric approach

A non-parametric approach

Our goal is to find a mapping $\phi : \mathcal{X} \to \mathbb{R}$ with a small test error

$$\mathcal{R}(\phi) := \mathbb{E}\big[(\phi(X) - Y)^2\big]$$

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

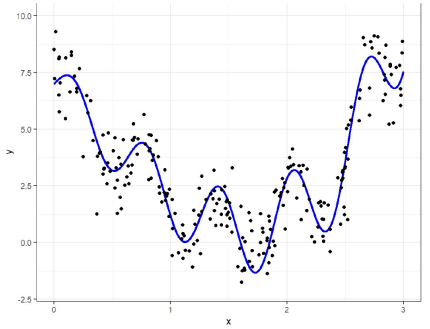
Approach 3: Apply a non-parametric technique based on the proximity of test feature vectors to training examples.

We want to approximate the function $\eta(x)$.

Idea: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$, then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

To estimate the conditional mean $\eta(x)$, we take the average of all Y_i satisfying that the associated X_i are close to x.

The k-nearest neighbour method



The k-nearest neighbour method

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

Idea: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$, then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

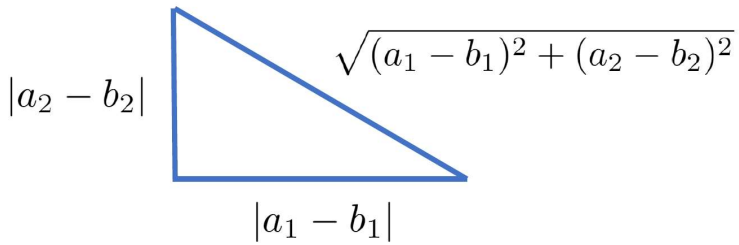
To estimate the conditional mean $\eta(x)$, we take the average of all Y_i satisfying that the associated X_i are close to x.

Step 1. Find k-nearest neighbours

Given a point $x \in \mathcal{X}$, we want to find k points from $(X_1, \cdots, X_n)$ with smallest distance to x.

We can measure the distance between vector $a = (a_1, \cdots, a_d) \in \mathbb{R}^d$ and $b = (b_1, \cdots, b_d) \in \mathbb{R}^d$ by the Euclidean norm:

$$\|a - b\|_2 = \sqrt{(a_1 - b_1)^2 + \cdots + (a_d - b_d)^2}$$



The k-nearest neighbour method

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

Idea: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$, then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

To estimate the conditional mean $\eta(x)$, we take the average of all Y_i satisfying that the associated X_i are close to x.

Step 1. Find k-nearest neighbours

Given a point $x \in \mathcal{X}$, we want to find k points from $(X_1, \cdots, X_n)$ with smallest distance to x.

We take a permutation (a reordering) $\tau_1(x), \cdots, \tau_n(x)$ of the set $\{1, \cdots, n\}$ with

$$\|x - X_{\tau_1(x)}\|_2 \leq \|x - X_{\tau_2(x)}\|_2 \leq \cdots \leq \|x - X_{\tau_n(x)}\|_2$$

Then $X_{\tau_1(x)}$, $X_{\tau_2(x)}$, $\cdots$, $X_{\tau_k(x)}$ are the k-nearest neighbour of x.

k is a hyper-parameter.

The k-nearest neighbour method

The optimal function (minimising $\mathcal{R}$) is $\eta(x) = \mathbb{E}(Y \mid X = x)$.

Idea: If $x_0 \in \mathcal{X}$ is close to $x_1 \in \mathcal{X}$, then we expect $\eta(x_0)$ to be close to $\eta(x_1)$.

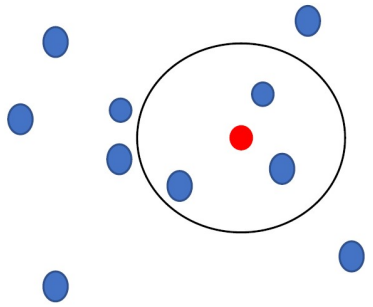
Suppose we have data $\mathcal{D} := ((X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n))$ where $(X_i, Y_i) \sim \mathbf{P}$

To estimate the conditional mean $\eta(x)$, we take the average of all Y_i satisfying that the associated X_i are close to x.

Step 1. Find k-nearest neighbours

Given a point $x \in \mathcal{X}$, we want to find k points from $(X_1, \cdots, X_n)$ with smallest distance to x.

$X_{\tau_1(x)}, X_{\tau_2(x)}, \cdots, X_{\tau_k(x)}$ are the k-nearest neighbours of x.



Step 2. Averaging

Set $\hat{\phi}_k(x) = \frac{1}{k} \sum_{j=1}^{k} Y_{\tau_j(x)}$

$\hat{\phi}(x)$ is the output of the regression model (can be viewed as an estimate of $\eta(x)$).

The k-nearest neighbour method in R

We can carry out the k-nearest neighbour method in R with "kknn"

```
library(kknn) # load knn library
```

Let's choose the number of nearest neighbours "k"

```
k<-15 # set number of neighbours
```

We can then train our k-nearest neighbour model.

```
knn_model<-train.kknn(y~.,data=train_data, ks = k, kernel = "rectangular")
```

This specifies the columns corresponding to the labels & features.

This data-data frame is our training dataset. The "y" column contains the labels.

We can now make predictions and estimate the performance.

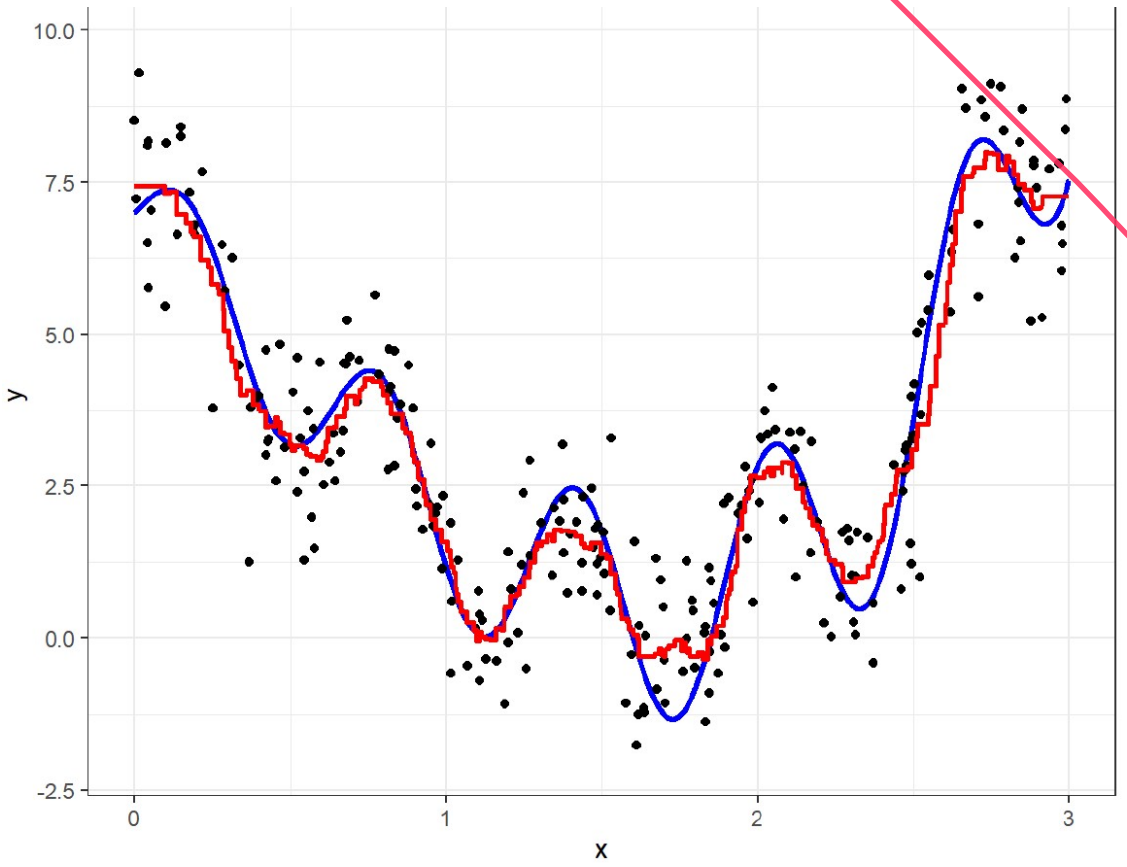
```
val_predictions<-predict(knn_model,val_x%>%data.frame()) # validation predictions
val_msq_error<-mean((val_predictions-val_y)^2)  # validation error
```

The k-nearest neighbour method

The k-nearest neighbour method gives a flexible approach to non-linear regression.

The approach is non-parametric meaning the complexity of the model grows with the sample size.

The number k is a hyper-parameter for regularisation.



(X_i, Y_i)

$(x, \eta(x))$

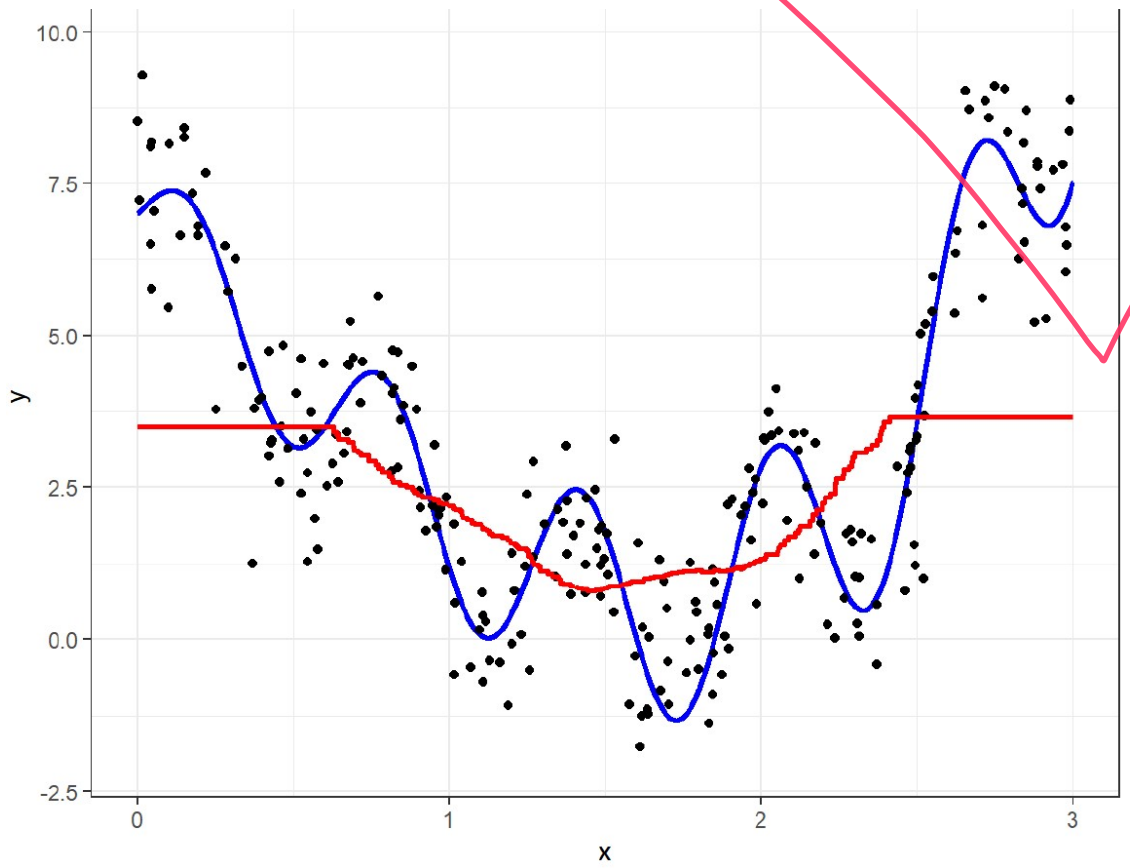
$(x, \hat{\phi}(x))$

number of neighbours $k = 15$.

The k-nearest neighbour method

If the number of neighbours k is too large, this method will underfit.

The method will have a very high bias and perform poorly on both training and test data.



(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

number of neighbours $k = 100$.

The k-nearest neighbour method

If the number of neighbours k is too small, this method will overfit.

The method will have a very low training error but a very high variance with high test error

(X_i, Y_i)

$(x, \eta(x))$

$(x, \hat{\phi}(x))$

number of neighbours $k = 1$.

What have we covered?

We investigated a small selection of different approaches for non-linear regression.

We began with an approach for applying linear methods to non-linear problems with random non-linear feature mappings.

- Hyper-parameters include the standard deviation of random features and norm penalty.

We then considered the neural network approach where the feature mapping is learnt from the data.

- Hyper-parameters include the weight decay as well as the structure of the network

We talked about the k-nearest neighbour method a flexible distance-based approach.

- Hyperparameters include the number of neighbours and the distance metric.

Each of these approaches can be applied to classification as well as regression.

University of BRISTOL

Thanks for listening!

Dr. Rihuan Ke

rihuan.ke@bristol.ac.uk

Statistical Computing and Empirical Methods
Unit EMATM0061, MSc Data Science