IAML: Linear Regression

Nigel Goddard
School of Informatics

Semester 1

Overview

- The linear model
- Fitting the linear model to data
- Probabilistic interpretation of the error function
- Examples of regression problems
- Dealing with multiple outputs
- Generalized linear regression
- Radial basis function (RBF) models

The Regression Problem

- Classification and regression problems:
  - Classification: target of prediction is discrete
  - Regression: target of prediction is continuous
- Training data: Set $\mathcal{D}$ of pairs $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$
- Today: Linear regression, i.e., relationship between $\mathbf{x}$ and y is linear.
- Although this is simple (and limited) it is:
  - More powerful than you would expect
  - The basis for more complex nonlinear methods
  - Teaches a lot about regression and classification

Examples of regression problems

- Robot inverse dynamics: predicting what torques are needed to drive a robot arm along a given trajectory
- Electricity load forecasting, generate hourly forecasts two days in advance (see W & F, §1.3)
- Predicting staffing requirements at help desks based on historical data and product and sales information,
- Predicting the time to failure of equipment based on utilization and environmental conditions

The Linear Model

- Linear model

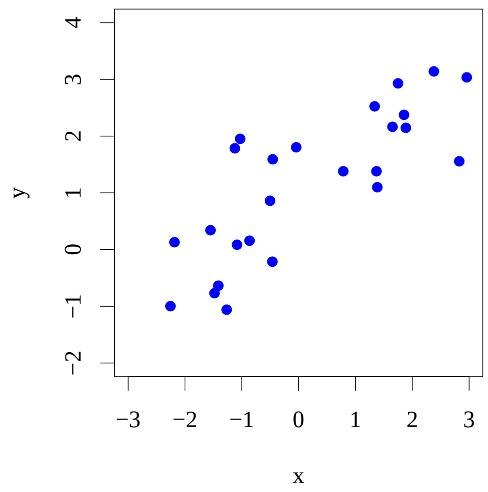
$$
\begin{aligned}
f(\mathbf{x};\mathbf{w}) &= w_0 + w_1 x_1 + \ldots + w_D x_D \\
&= \phi(\mathbf{x})\mathbf{w}
\end{aligned}
$$

where $\phi(\mathbf{x}) = (1, x_1, \ldots, x_D) = (1, \mathbf{x}^T)$
and

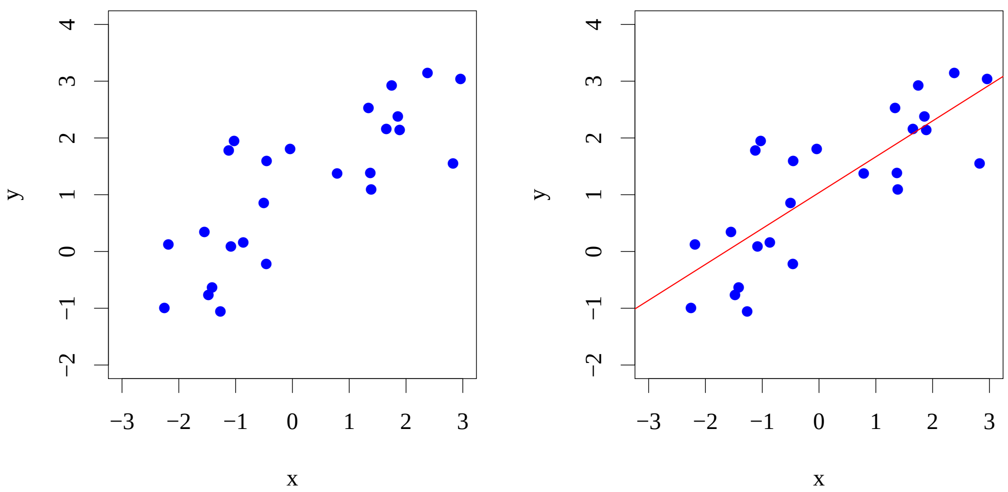
$$
\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \ldots \\ w_D \end{pmatrix} \tag{1}
$$

- The maths of fitting linear models to data is easy. We use the notation $\phi(\mathbf{x})$ to make generalisation easy later.

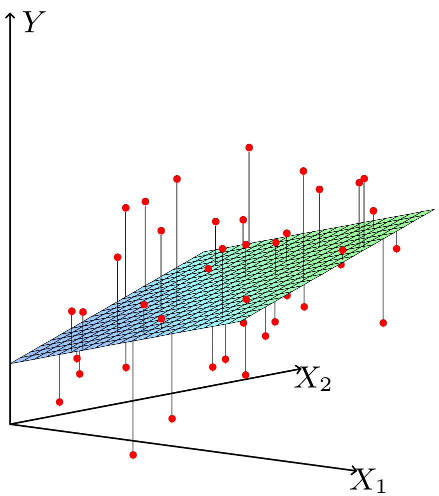
Toy example: Data



Toy example: Data



With two features



Instead of a line, a *plane*. With more features, a *hyperplane*.

Figure: Hastie, Tibshirani, and Friedman

With more features

CPU Performance Data Set

- Predict: PRP: published relative performance
- MYCT: machine cycle time in nanoseconds (integer)
- MMIN: minimum main memory in kilobytes (integer)
- MMAX: maximum main memory in kilobytes (integer)
- CACH: cache memory in kilobytes (integer)
- CHMIN: minimum channels in units (integer)
- CHMAX: maximum channels in units (integer)

With more features

```
PRP = - 56.1
      + 0.049 MYCT
      + 0.015 MMIN
      + 0.006 MMAX
      + 0.630 CACH
      - 0.270 CHMIN
      + 1.46 CHMAX
```


In matrix notation

- Design matrix is $n \times (D+1)$

$$\Phi = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nD} \end{pmatrix}$$

- x_{ij} is the jth component of the training input $\mathbf{x}_i$
- Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- Then $\hat{\mathbf{y}} = \Phi\mathbf{w}$ is ...?

Linear Algebra: The 1-Slide Version

What is matrix multiplication?

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

First consider matrix times vector, i.e., $A\mathbf{b}$. Two answers:

1. $A\mathbf{b}$ is a linear combination of the columns of A

$$Ab = b_1 \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} + b_2 \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix} + b_3 \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}$$

2. $A\mathbf{b}$ is a vector. Each element of the vector is the dot products between b and *one row* of A.

$$A\mathbf{b} = \begin{pmatrix} (a_{11}, a_{12}, a_{13})\mathbf{b} \\ (a_{21}, a_{22}, a_{23})\mathbf{b} \\ (a_{31}, a_{32}, a_{33})\mathbf{b} \end{pmatrix}$$

Linear model (part 2)

In matrix notation:

- Design matrix is $n \times (D+1)$

$$\Phi = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nD} \end{pmatrix}$$

- x_{ij} is the jth component of the training input $\mathbf{x}_i$
- Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- Then $\hat{\mathbf{y}} = \Phi\mathbf{w}$ is the model's predicted values on training inputs.

Solving for Model Parameters

This looks like what we've seen in linear algebra

$$\mathbf{y} = \Phi\mathbf{w}$$

We know $\mathbf{y}$ and Φ but not $\mathbf{w}$.

So why not take $\mathbf{w} = \Phi^{-1}\mathbf{y}$? (You can't, but why?)

Solving for Model Parameters

This looks like what we've seen in linear algebra

$$\mathbf{y} = \Phi\mathbf{w}$$

We know $\mathbf{y}$ and Φ but not $\mathbf{w}$.

So why not take $\mathbf{w} = \Phi^{-1}\mathbf{y}$? (You can't, but why?)

Three reasons:

- Φ is not square. It is $n \times (D+1)$.
- The system is overconstrained (n equations for $D+1$ parameters), in other words
- The data has noise

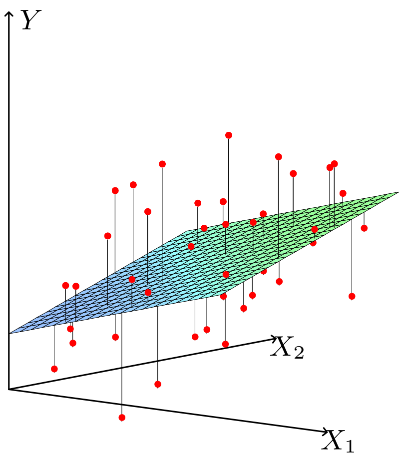
Loss function

Want a *loss function* $O(\mathbf{w})$ that

- We minimize wrt $\mathbf{w}$.
- At minimum, $\hat{\mathbf{y}}$ looks like $\mathbf{y}$.
- (Recall: $\hat{\mathbf{y}}$ depends on $\mathbf{w}$)

$$\hat{\mathbf{y}} = \Phi\mathbf{w}$$

Fitting a linear model to data



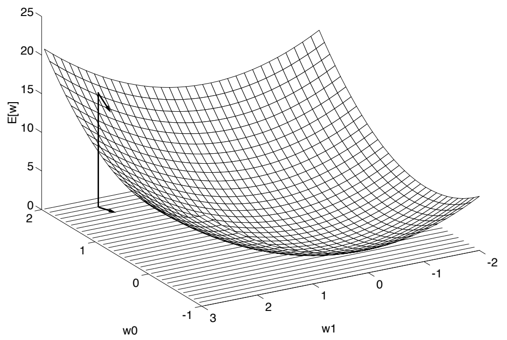
- A common choice: *squared error* (makes the maths easy)

$$O(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- In the picture: this is sum of squared length of black sticks.
- (Each one is called a *residual*, i.e., each $y_i - \mathbf{w}^T \mathbf{x}_i$)

Fitting a linear model to data

- $$O(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$
  $$= (\mathbf{y} - \Phi\mathbf{w})^T (\mathbf{y} - \Phi\mathbf{w})$$
- We want to minimize this with respect to $\mathbf{w}$.
- The error surface is a parabolic bowl



- How do we do this?

The Solution

- Answer: to minimize $O(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$, set partial derivatives to 0.
- This has an analytical solution

$$\hat{\mathbf{w}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

- $(\Phi^T \Phi)^{-1} \Phi^T$ is the pseudo-inverse of Φ
- First check: Does this make sense? Do the matrix dimensions line up?
- Then: Why is this called a pseudo-inverse? ()
- Finally: What happens if there are no features?

Probabilistic interpretation of $O(\mathbf{w})$

- Assume that $y = \mathbf{w}^T \mathbf{x} + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$
- (This is an exact linear relationship plus Gaussian noise.)
- This implies that $y | \mathbf{x}_i \sim N(\mathbf{w}^T \mathbf{x}_i, \sigma^2)$, i.e.

$$-\log p(y_i | \mathbf{x}_i) = \log \sqrt{2\pi} + \log \sigma + \frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2}$$

- So minimising $O(\mathbf{w})$ equivalent to maximising likelihood!
- Can view $\mathbf{w}^T \mathbf{x}$ as $E[y | \mathbf{x}]$.
- Squared residuals allow estimation of σ^2

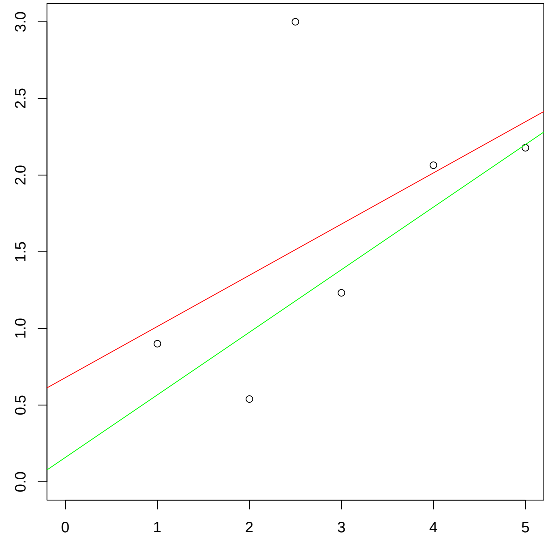
$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Fitting this into the general structure for learning algorithms:

- Define the **task**: regression
- Decide on the **model structure**: linear regression model
- Decide on the **score function**: squared error (likelihood)
- Decide on **optimization/search method** to optimize the score function: calculus (analytic solution)

Sensitivity to Outliers

- Linear regression is sensitive to *outliers*
- Example: Suppose $y = 0.5x + \epsilon$, where $\epsilon \sim N(0, \sqrt{0.25})$, and then add a point (2.5,3):



Diagnositics

Graphical diagnostics can be useful for checking:

- Is the relationship obviously nonlinear? Look for structure in residuals?
- Are there obvious outliers?

The goal isn't to find all problems. You can't. The goal is to find obvious, embarrassing problems.

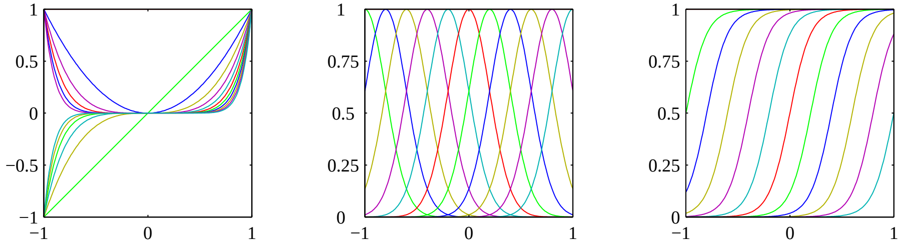
Examples: Plot residuals by fitted values. Stats packages will do this for you.

Dealing with multiple outputs

- Suppose there are q different targets for each input $\mathbf{x}$
- We introduce a different $\mathbf{w}_i$ for each target dimension, and do regression separately for each one
- This is called *multiple regression*

Basis expansion

- We can easily transform the original attributes $\mathbf{x}$ non-linearly into $\phi(\mathbf{x})$ and do linear regression on them



polynomial Gaussians sigmoids

Figure credit: Chris Bishop, PRML

- Design matrix is $n \times m$

$$\Phi = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \dots & \phi_m(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \dots & \phi_m(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\mathbf{x}_n) & \phi_2(\mathbf{x}_n) & \dots & \phi_m(\mathbf{x}_n) \end{pmatrix}$$

- Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- Minimize $E(\mathbf{w}) = |\mathbf{y} - \Phi\mathbf{w}|^2$. As before we have an analytical solution

$$\hat{\mathbf{w}} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{y}$$

- $(\Phi^T\Phi)^{-1}\Phi^T$ is the pseudo-inverse of Φ

Example: polynomial regression

$\phi(x) = (1,\ x,\ x^2, \dots, x^M)^T$

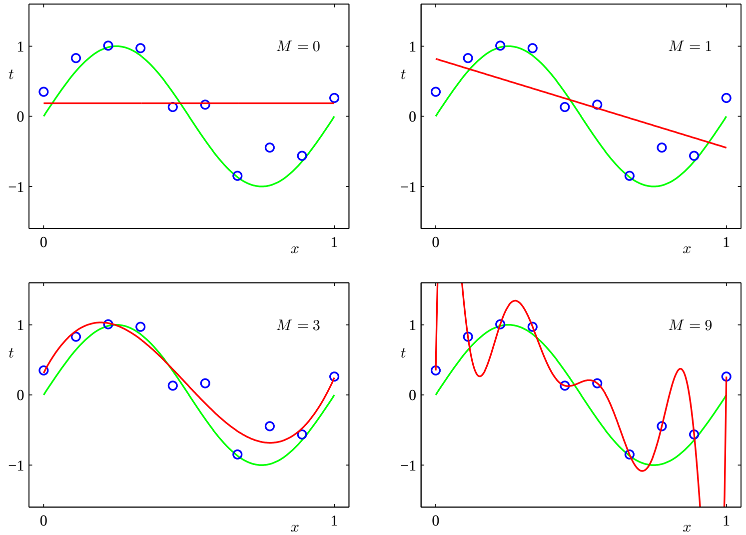


Figure credit: Chris Bishop, PRML

More about the features

- Transforming the features can be important.
- Example: Suppose I want to predict the CPU performance.
- Maybe one of the features is *manufacturer*.

$$x_1 = \begin{cases} 1 & \text{if Intel} \\ 2 & \text{if AMD} \\ 3 & \text{if Apple} \\ 4 & \text{if Motorola} \end{cases}$$

- Let's use this as a feature. Will this work?

More about the features

- Transforming the features can be important.
- Example: Suppose I want to predict the CPU performance.
- Maybe one of the features is *manufacturer*.

$$x_1 = \begin{cases} 1 & \text{if Intel} \\ 2 & \text{if AMD} \\ 3 & \text{if Apple} \\ 4 & \text{if Motorola} \end{cases}$$

- Let's use this as a feature. Will this work?
- Not the way you want. Do you really believe AMD is double Intel?

More about the features

- Instead convert this into 0/1

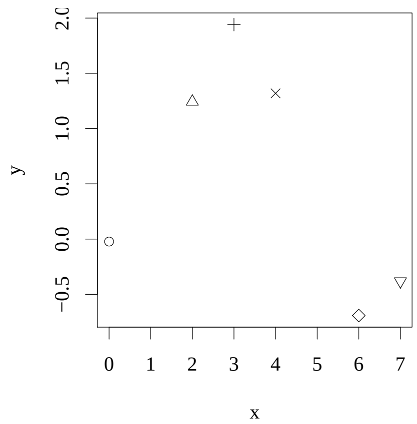
$$\begin{aligned} x_1 &= 1 \text{ if Intel, 0 otherwise} \\ x_2 &= 1 \text{ if AMD, 0 otherwise} \\ &\vdots \end{aligned}$$

- Note this is a consequence of linearity. We saw something similar with text in the first week.
- Other good transformations: log, square, square root

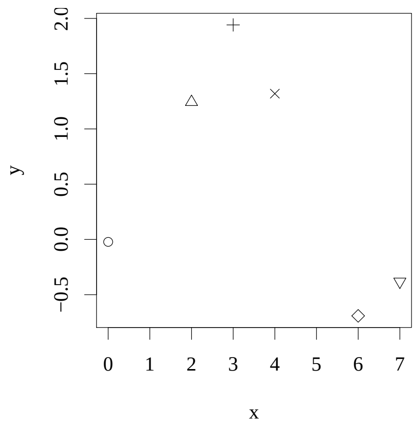
Radial basis function (RBF) models

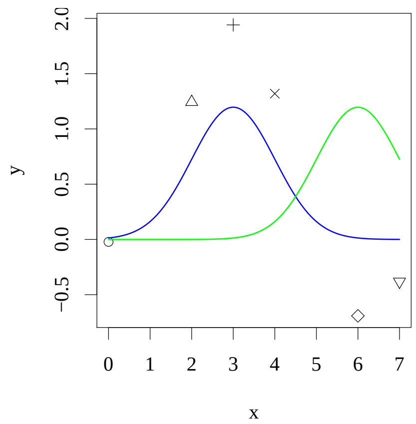
- Set $\phi_i(\mathbf{x}) = \exp(-\frac{1}{2}|\mathbf{x} - \mathbf{c}_i|^2/\alpha^2)$.
- Need to position these "basis functions" at some prior chosen centres $\mathbf{c}_i$ and with a given width α. There are many ways to do this but choosing a subset of the datapoints as centres is one method that is quite effective
- Finding the weights is the same as ever: the pseudo-inverse solution.

RBF example



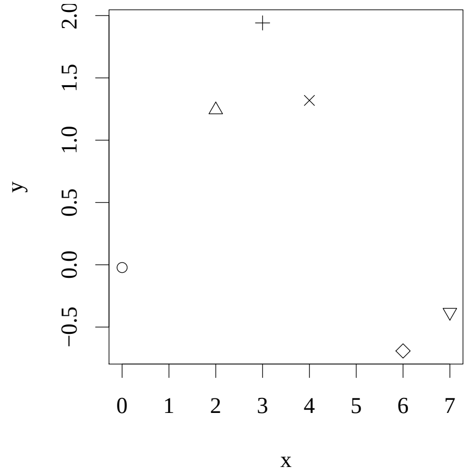
RBF example



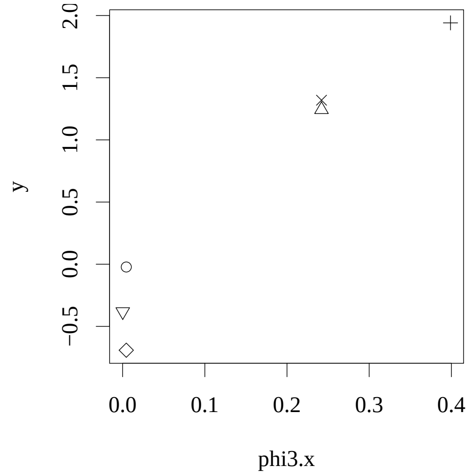


An RBF feature

Original data



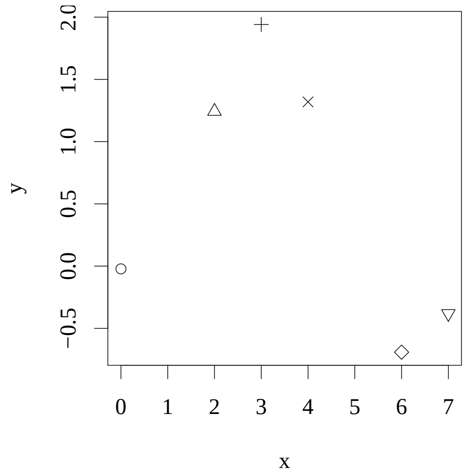
RBF feature, $c_1 = 3$, $\alpha_1 = 1$



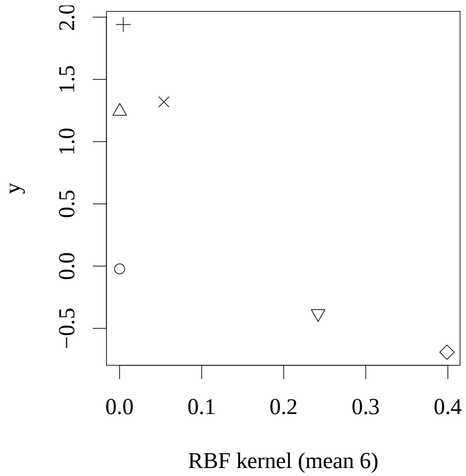
Another RBF feature

Notice how the feature functions “specialize” in input space.

Original data



RBF feature, $c_2 = 6$, $\alpha_2 = 1$

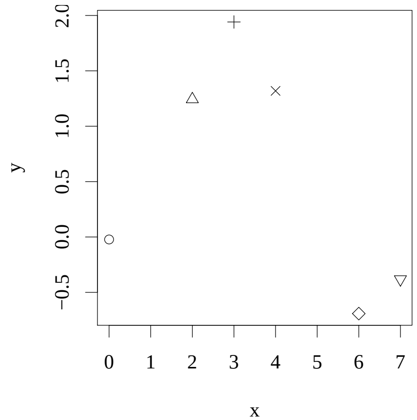


RBF example

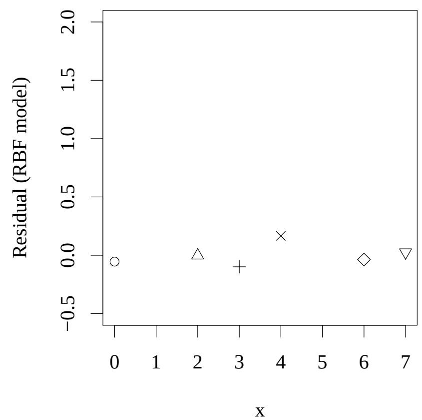
Run the RBF with both basis functions above, plot the residuals

$$y_i - \phi(\mathbf{x}_i)^T \mathbf{w}$$

Original data



Residuals



RBF: Ay, there's the rub

- So why not use RBFs for everything?
- Short answer: You might need too many basis functions.
- This is especially true in high dimensions (we'll say more later)
- Too many means you probably overfit.
- Extreme example: Centre one on each training point.
- Also: notice that we haven't seen yet in the course how to learn the RBF *parameters*, i.e., the mean and standard deviation of each kernel
- Main point of presenting RBFs now: Set up later methods like support vector machines

Summary

- Linear regression often useful out of the box.
- More useful than it would be seem because linear means linear *in the parameters*. You can do a nonlinear transform of the data first, e.g., polynomial, RBF. This point will come up again.
- Maximum likelihood solution is computationally efficient (pseudo-inverse)
- Danger of overfitting, especially with many features or basis functions