CO902
Probabilistic and statistical inference

Lecture 5

Tom Nichols
Department of Statistics &
Warwick Manufacturing Group

t.e.nichols@warwick.ac.uk

Admin

- Project ("Written assignment")
  - Posted last Wednesday, followed by email
  - Binary classification based on 70-dimensional binary data
  - Work in pairs (maybe 1 group of 3), produce individual write up
    - No more than 4 sides A4
    - Use scientific style; see details on webpage
  - Please notify me of pairs (see spread sheet)
    - Balance-out Matlab expertise (if you're shaky, find a power-user)
  - Due date: 9AM Monday 11 February
    - But, will accept them for full credit until Noon Wednesday 13 February
  - Questions?
- Presentation ("Critical Reading Assignment")
  - 10 minute presentation, 25 Feb & 4 Mar
  - Based on scientific article that uses machine learning
    - … more next week
- Wrap up on discriminant analysis… (Lecture 4)

Outline of course

A. Basics: Probability, random variables (RVs), common distributions, introduction to statistical inference

B. Supervised learning: Classification, regression; including issues of over-fitting; penalized likelihood & Bayesian approaches

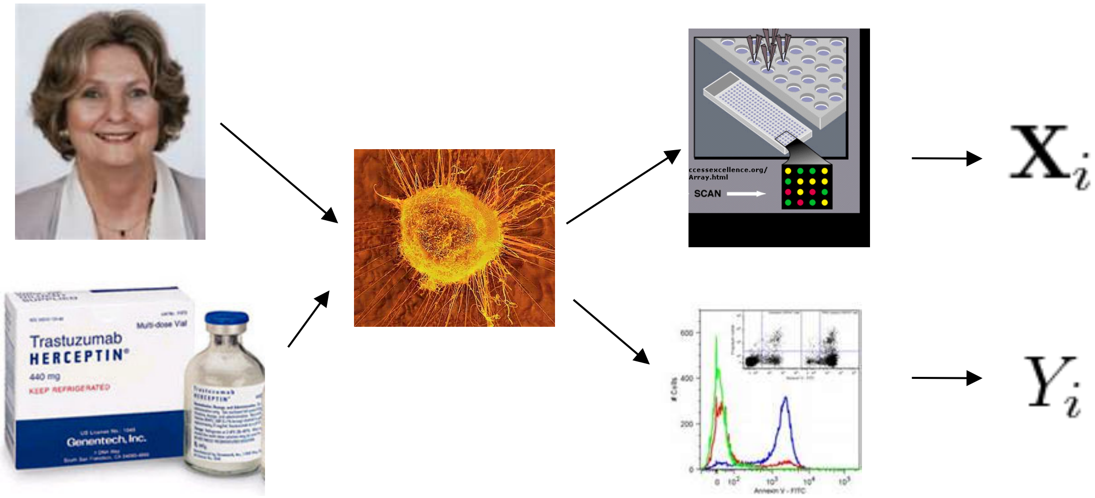
C. Unsupervised learning: Dimensionality reduction, clustering and mixture models

D. Networks: Probabilistic graphical models, learning in graphical models, inferring network structure

Today

- Probabilistic view of regression
- Over-fitting in regression
- Penalized likelihood: “ridge regression”
- Bayesian regression

Predicting drug response



- Suppose we collect **data** of the following kind:
  - For each of *n* patients, we get a tumour sample, and using a microarray obtain **expression measurements** for d=10k genes
  - Also, we administer the drug to each of the *n* patients, and record a quantitative measure of **drug response**
- This gives us data of the following kind:

$$\{\mathbf{X}_i, Y_i\},\ i = 1..n$$

$$\mathbf{X}_i \in \mathbb{R}^d,\ Y_i \in \mathbb{R}$$

Classification and regression

- **Supervised learning:** prediction problems where you start with a dataset in which the "right" answers are given
- Supervised in the sense of "learning with a teacher"

$$\{\mathbf{X}_i, Y_i\},\ i = 1..n$$

$\mathbf{X}_i \in \mathbb{R}^d,\ Y_i \in \{1, 2, \ldots k\}$	$\mathbf{X}_i \in \mathbb{R}^d,\ Y_i \in \mathbb{R}$
Classification	**Regression**

- Classification and regression are closely related (e.g. classifiers we've seen can be viewed as a type of regression called logistic regression)

Regression

- **Regression:** predicting real-valued outputs *Y* from inputs *X*
- In other words: supervised learning with quantitative rather than categorical outputs

- Recent decades have seen much progress in understanding:
  - Statistical aspects: accounting for random variation in data, learning parameters etc.
  - Practical aspects: empirically evaluating predictive ability etc.

- But open questions abound, e.g.:
  - Interplay between predictors
  - High-dimensional input spaces
  - Sparse prediction

Linear regression

- Simplest function: *Y* is a linear combination of components of vector *X*:

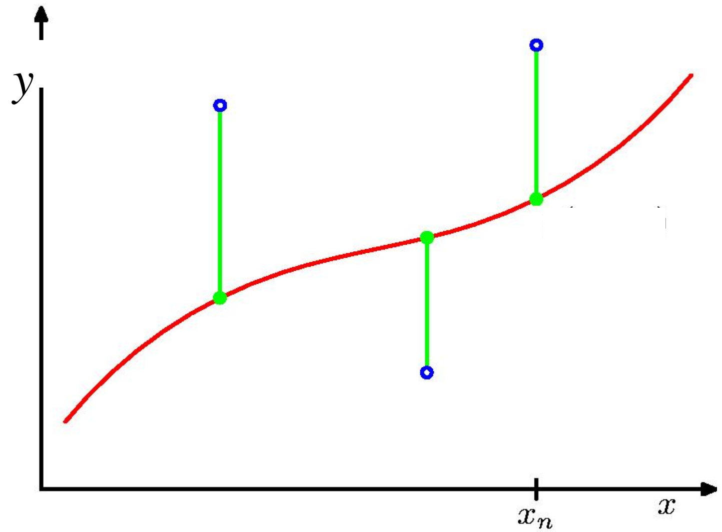
$$
\begin{aligned}
\hat{Y}(\mathbf{X}, \mathbf{w}) &= w_0 + w_1 X_1 + w_2 X_2 \ldots w_d X_d \\
&= \mathbf{w}^T \tilde{\mathbf{X}} \\
\mathbf{w} &= [w_0\ w_1 \ldots w_d]^T \\
\mathbf{X} &= [1\ X_1\ X_2 \ldots X_d]^T
\end{aligned}
$$

- Here, parameters are the “weights” ***w***
- To start with, we'd like to choose ***w*** such that the predictions fit the data well

Residual sum of squares

- **Residual sum of squares** captures the difference between the *n* predictions and corresponding true output values:

$$
\begin{aligned}
J(\mathbf{w}) &= \sum_{i=1}^{n}(Y_i - \mathbf{w}^T\mathbf{X}_i)^2 \\
&= \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2 \\
\mathbf{X} &= [\mathbf{X}_1 \ldots \mathbf{X}_n]^T \\
\mathbf{Y} &= [Y_1 \ldots Y_n]^T
\end{aligned}
$$



- Matrix ***X*** is *n* by *(d+1)*, it's just all of the input data together
  - Sometimes called the "design matrix"
- Components of vector *Y* are the *n* (true) outputs

Matrix notation

- Sum of squares in matrix notation:

$$J(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2$$

- We want:

$$\begin{aligned}\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}}\, \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2\end{aligned}$$

- This is now simply a problem in linear algebra
- **Q: what combination of the columns of *X* bring us closest to *Y*, or what are the co-ordinates of the projection of *Y* onto the column space of X?**

Solution

- Learn parameters to minimize residual sum of squares:

$$\begin{aligned}\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}}\, \|\mathbf{X}\mathbf{w} - \mathbf{Y}\|^2\end{aligned}$$

- Solution given by:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

- But, much safer to use the Moore-Penrose pseudo-inverse

$$\hat{\mathbf{w}} = \mathbf{X}^{-}\mathbf{Y}$$

  - "pinv(X)" in Matlab
  - Numerically stable
  - Gives one (of infinite number) of solutions if X rank deficient

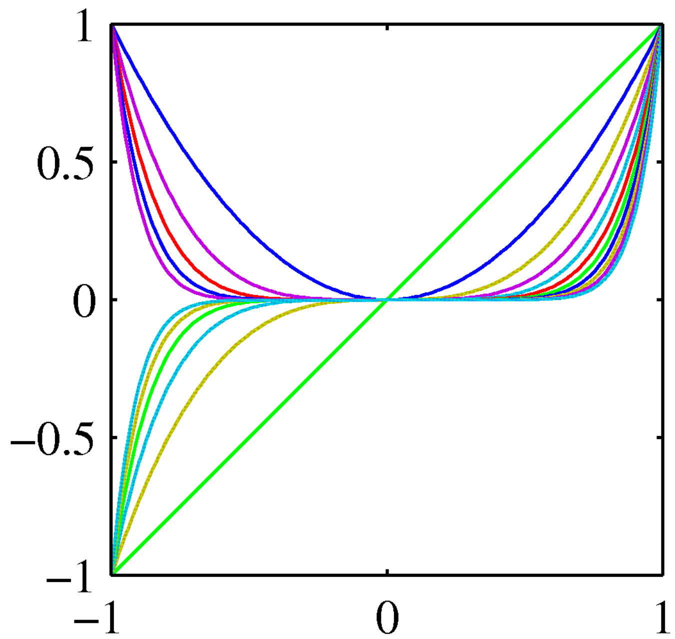
Polynomial regression

- This was entirely linear
- We can extend this approach by allowing the data to pass through a set of functions

Polynomial regression

- Prediction function (for now, assume *X* scalar):

$$
\begin{aligned}
\hat{Y}(X,\mathbf{w}) &= w_0 + w_1 X + w_2 X^2 \ldots w_k X^k \\
&= \mathbf{w}^T \phi(X) \\
\mathbf{w} &= [w_0 \; w_1 \ldots w_k]^T \\
\phi(X) &= [1 \; X \; X^2 \ldots X^k]^T
\end{aligned}
$$



- Residual sum of squares:

$$
\begin{aligned}
J(\mathbf{w}) &= \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \phi(X_i))^2 \\
&= \|\mathbf{\Phi w} - \mathbf{Y}\|^2 \\
\mathbf{\Phi} &: n \times (k+1) \\
\mathbf{Y} &= [Y_1 \ldots Y_n]^T
\end{aligned}
$$

Polynomial regression

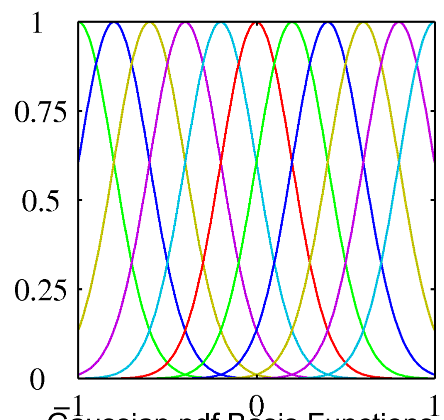
- Least squares solution:

$$
\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w}) \\
&= \underset{\mathbf{w}}{\operatorname{argmin}}\, \|\mathbf{\Phi}\mathbf{w} - \mathbf{Y}\|^2 \\
&= (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{Y} \\
&= \mathbf{\Phi}^{-}\mathbf{Y}
\end{aligned}
$$

Regression using basis functions

- More generally, we can think of transforming input data using *k* basis functions (R^d *to* R), linear regression is then a special case:

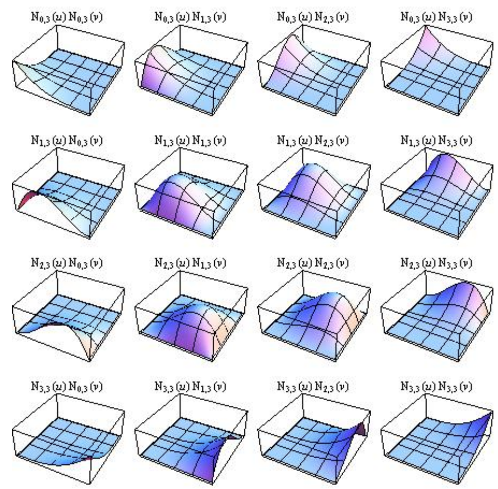
$$
\begin{aligned}
\hat{Y}(\mathbf{X}, \mathbf{w}) &= w_0 + w_1\phi_1(\mathbf{X}) + w_2\phi_2(\mathbf{X}) \ldots w_k\phi_k(\mathbf{X}) \\
&= \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}) \\
\mathbf{w} &= [w_0\ w_1 \ldots w_k]^T \\
\boldsymbol{\phi}(\mathbf{X}) &= [1\ \phi_1(\mathbf{X}) \ldots \phi_k(\mathbf{X})]^T
\end{aligned}
$$



Gaussian pdf Basis Functions

- In a similar fashion to simple linear and polynomial regression this gives:

$$
\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i))^2 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \|\boldsymbol{\Phi}\mathbf{w} - \mathbf{Y}\|^2 \\
\boldsymbol{\Phi} &: n \times (k+1) \\
\mathbf{Y} &= [Y_1 \ldots Y_n]^T
\end{aligned}
$$


2D Spline Basis Functions

Regression using basis functions

- The least-squares solution is obtained using the pseudo-inverse of the design matrix:

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

- Same as before because it's still linear in the parameters, despite non-linear functions of X

A probability model

- Nothing we've seen so far is a **probability model**
- We can couch linear regression in probabilistic terms by considering the conditional distribution of output *Y given* input vector ***X*** and parameters:

$$p(Y \mid \mathbf{X}, \mathbf{w}, \boldsymbol{\theta})$$

- We get here by a similar argument to the one we used for classification, starting from P(X,Y|w,\theta)

A probabilistic model

- Conditional distribution of output *Y given* input vector ***X*** and parameters:

$$p(Y \mid \mathbf{X}, \mathbf{w}, \boldsymbol{\theta})$$

- This is a density over Y, which tells us how *Y* varies given a specific observation of ***X***
- The parameters include the weights for the prediction function, but also includes other parameters
- We'll assume the conditional distribution is a **Normal...**

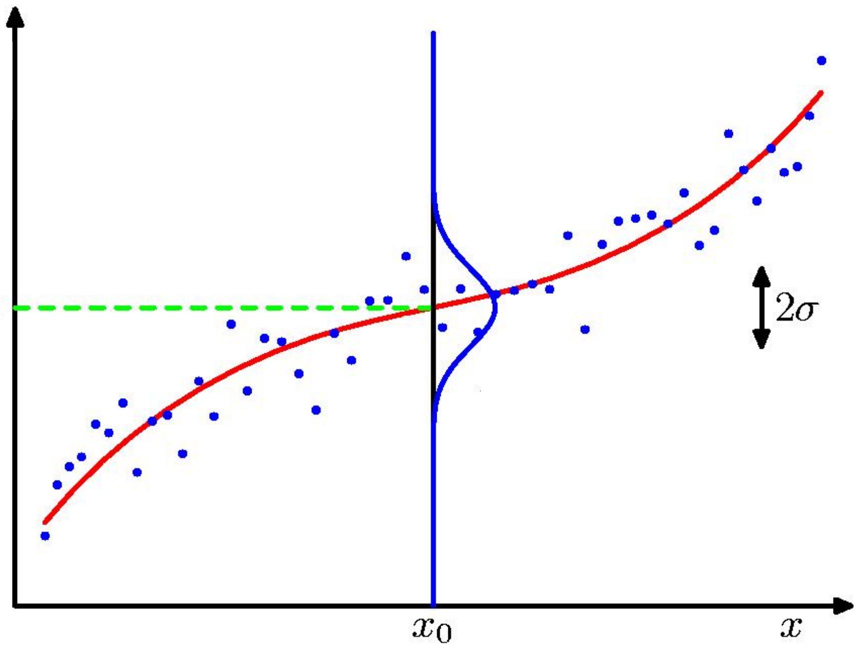
Normal model

- Normal model:

$$p(Y \mid \mathbf{X}, \mathbf{w}, \sigma^2) \;\; = \;\; \mathcal{N}(Y \mid \mathbf{w}^T \phi(\mathbf{X}), \sigma^2)$$

- This tells us that given ***X***, *Y's* distribution is a **Normal pdf**, centred on the output we'd get using the inputs ***X*** and weights ***w***
  - A *conditional* model
- Can also be written as

  output = deterministic part + noise



Likelihood function

- Assuming outputs are independent given inputs (or "conditionally independent"), we get the following likelihood:

$$p(Y_1 \ldots Y_n \mid \mathbf{X}_1 \ldots \mathbf{X}_n, \mathbf{w}, \sigma^2) \quad = \quad \prod_{i=1}^{n} \mathcal{N}(Y_i \mid \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i), \sigma^2)$$

- Now we're in a position to **estimate** the weights ***w***

- **Q: Using the likelihood function above, what's the Maximum likelihood estimate of *w*?**

Log-likelihood

- Log-likelihood:

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{n} log\left(\frac{1}{\sqrt{2\pi}\sigma^2}\right) - \frac{1}{2}\left(\frac{Y_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{X}_i)}{\sigma^2}\right)^2 \\
&= const - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{X}_i))^2
\end{aligned}
$$

MLE

- MLE:

$$
\begin{aligned}
\hat{\mathbf{w}}_{MLE} &= \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i))^2 \\
&= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i))^2 \\
&= \arg\min_{\mathbf{w}} \|\boldsymbol{\Phi}\mathbf{w} - \mathbf{Y}\|^2
\end{aligned}
$$

- This gives $\hat{\mathbf{w}}_{MLE} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{Y}$

- Thus, due to the quadratic term in the Normal exponent, the **MLE under a Normal model is identical to the least-squares solution**

Polynomial regression: example

- Prediction function (for now, assume *X* scalar):

$$
\begin{aligned}
\hat{Y}(X, \mathbf{w}) &= w_0 + w_1 X + w_2 X^2 \ldots w_k X^k \\
&= \mathbf{w}^T \boldsymbol{\phi}(X) \\
\mathbf{w} &= [w_0 \; w_1 \ldots w_k]^T \\
\boldsymbol{\phi}(X) &= [1 \; X \; X^2 \ldots X^k]^T
\end{aligned}
$$

- Residual sum of squares:

$$
\begin{aligned}
J(\mathbf{w}) &= \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \boldsymbol{\phi}(X_i))^2 \\
&= \|\mathbf{\Phi}\mathbf{w} - \mathbf{Y}\|^2 \\
\mathbf{\Phi} &: \; n \times (k+1) \\
\mathbf{Y} &= [Y_1 \ldots Y_n]^T
\end{aligned}
$$

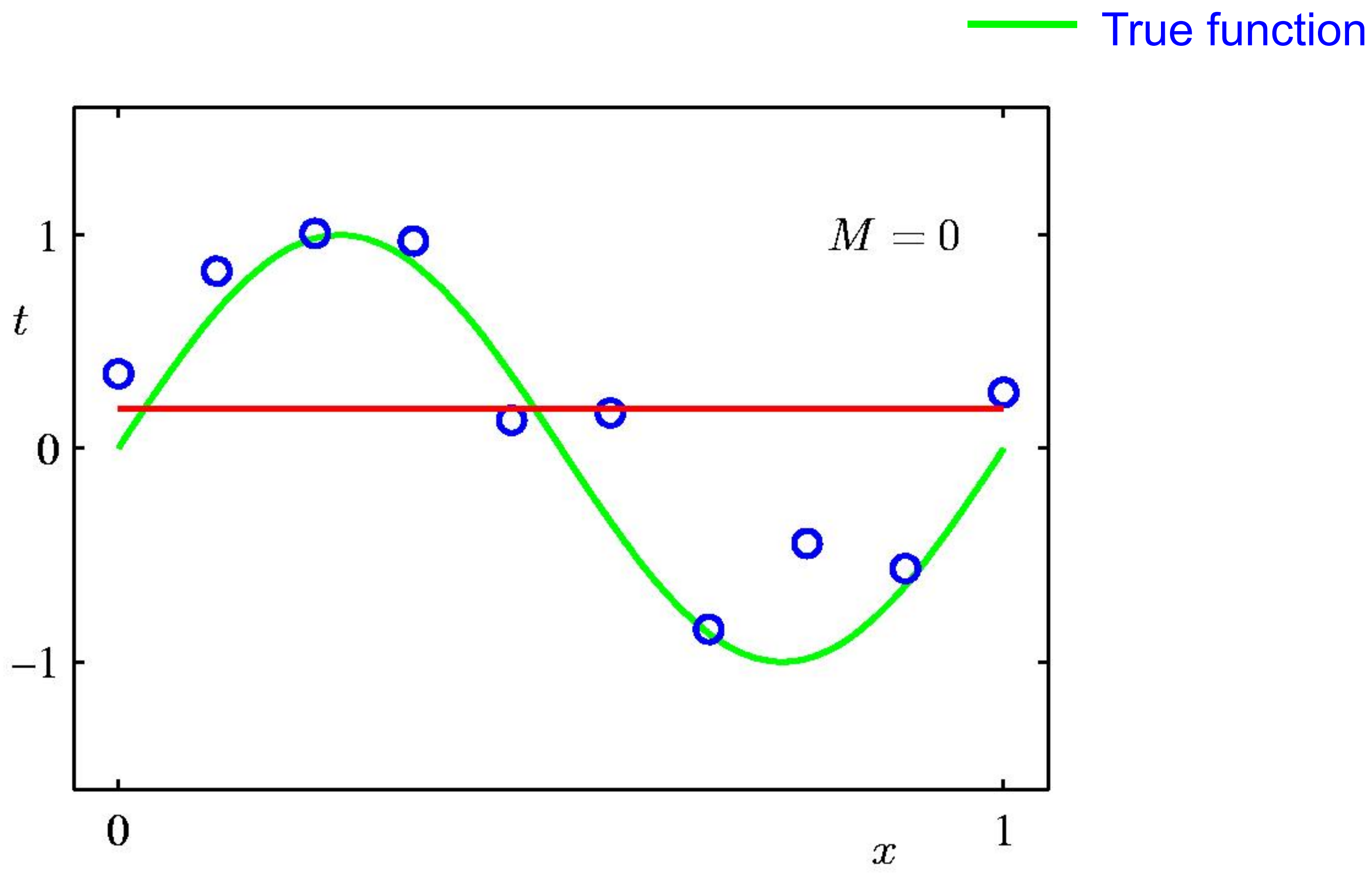
Polynomial regression: example

- Least squares solution:

$$
\begin{aligned}
\hat{\mathbf{w}} &= \operatorname*{argmin}_{\mathbf{w}} J(\mathbf{w}) \\
&= \operatorname*{argmin}_{\mathbf{w}} \|\mathbf{\Phi}\mathbf{w} - \mathbf{Y}\|^2 \\
&= (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{Y} \\
&= \mathbf{\Phi}^{-}\mathbf{Y}
\end{aligned}
$$

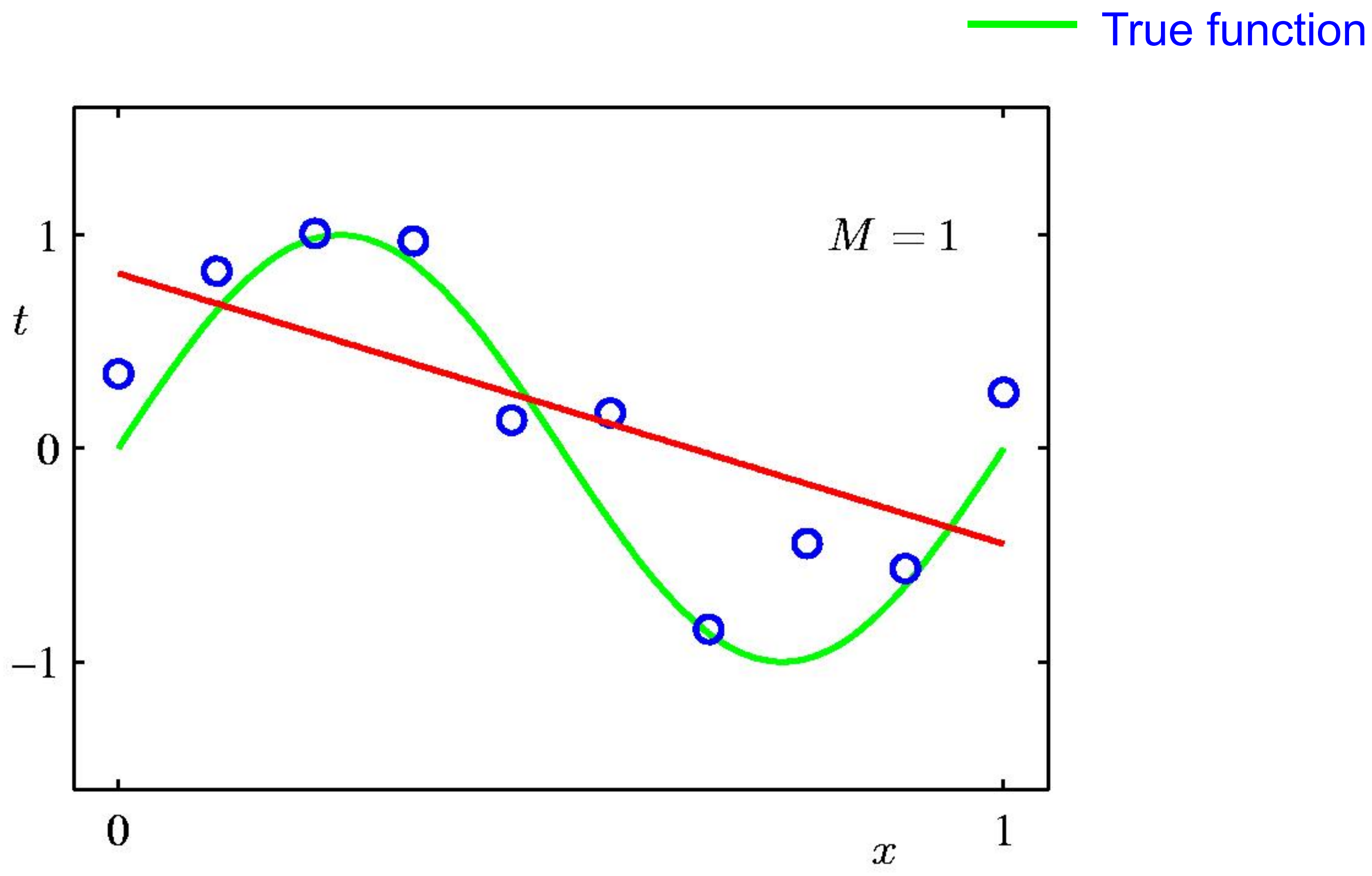
Example: order k polynomial

- **k=0**



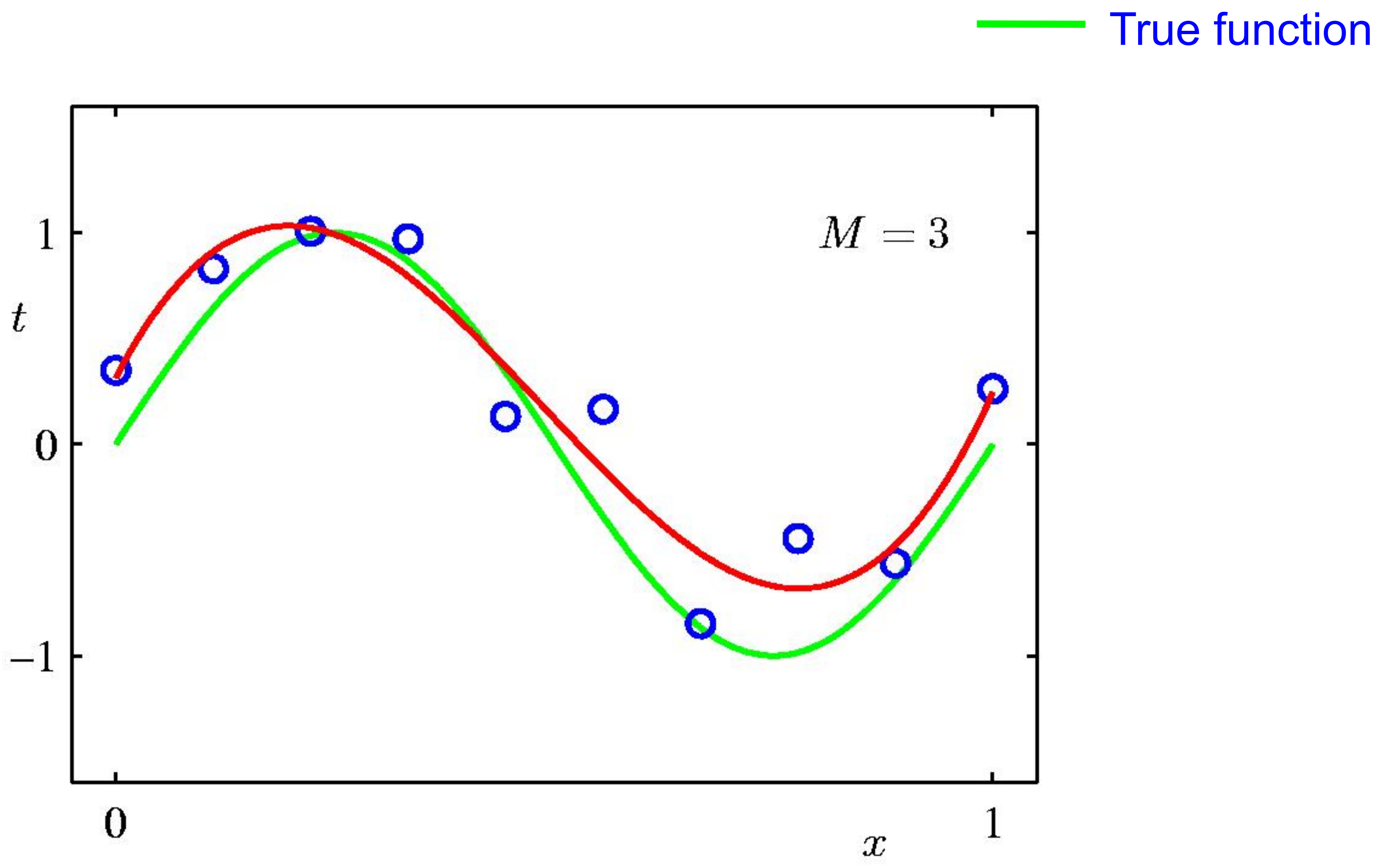
Example: order k polynomial

- **k=1**



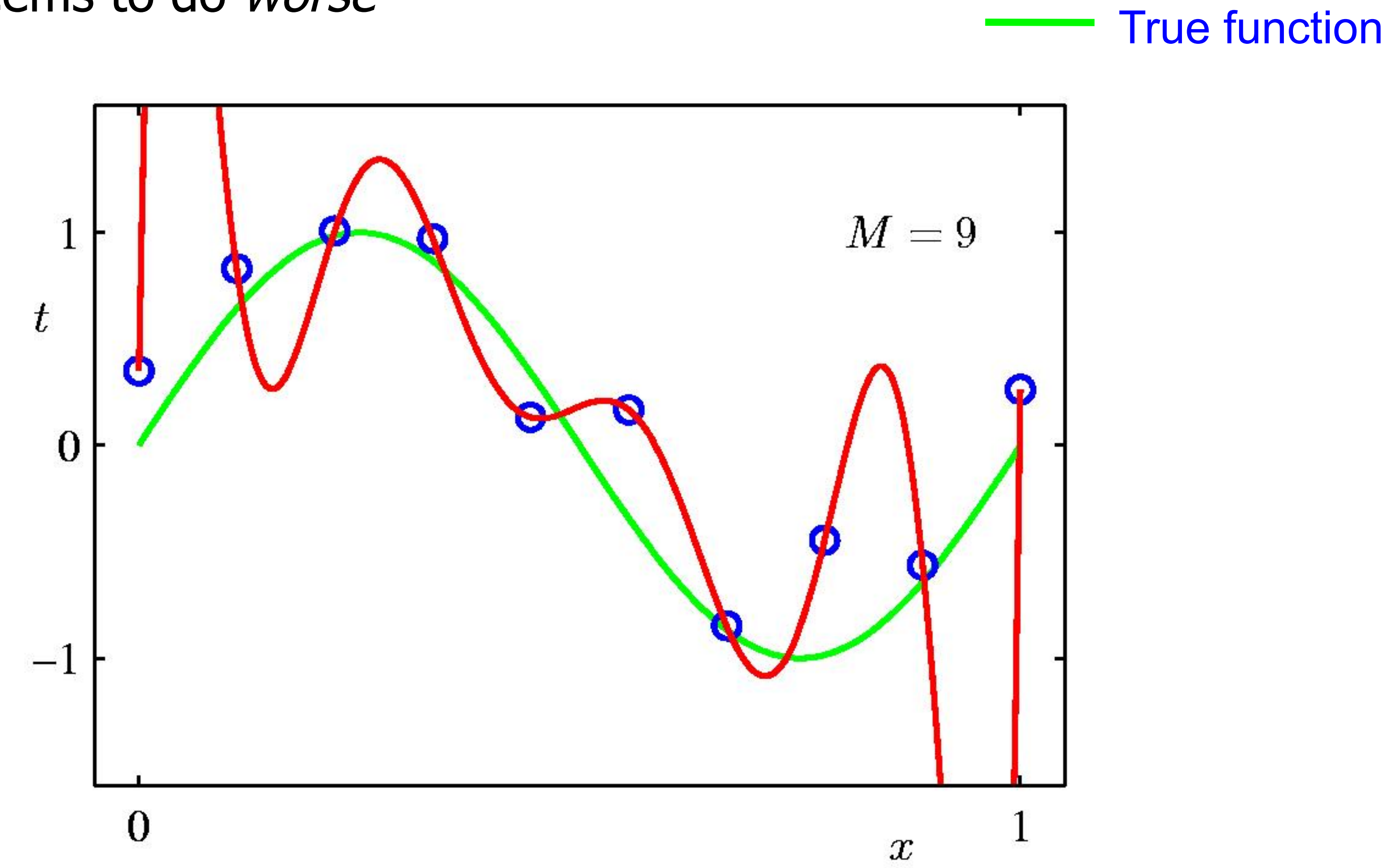
Example: order k polynomial

- **k=3**

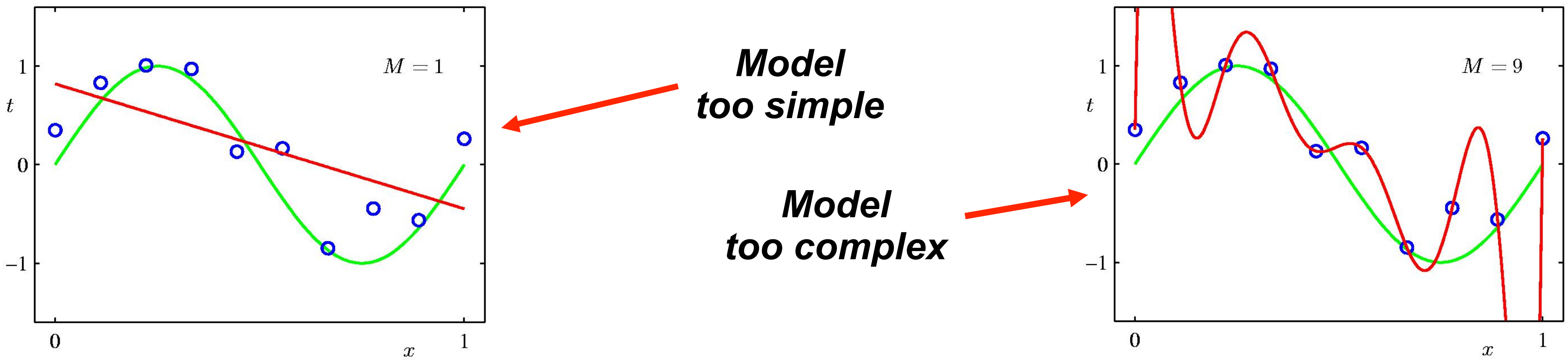


Example: order k polynomial

- **k=9**
  - k=9 subsumes k=3, in that sense it's more powerful, more general
  - But seems to do *worse*



Model complexity



- Closely fitting a complex model to the data may not be predictive!
- This is an example of **overfitting**

- We have to be
  - Careful about the choice of prediction function:
    - if it's too general, we run the risk of overfitting (e.g. k=9)
    - if it's too restricted we may not be able to capture the relationship between input and output (e.g. k=1)
  - If we *do* use relatively complex models, with many parameters, we must be careful about learning the parameters

Model selection

- So we *have* to negotiate a trade-off and choose a good level of model complexity – but how?

- This is a problem in **model selection**, it can be done:

  – Using Bayesian methods,

  – By augmenting the likelihood the to penalize complex models

  – Empirically, e.g. using test data, or cross-validation

Train and test paradigm

- Recall **“train and test”** idea from classification
- Idea: since we're interested in **predictive ability** on unseen data, why not “train” on a *subset* of the data and “test” on the remainder?
- This would give us some indication of how well we'd be likely to do on new data...
- These “train and test” curves have a characteristic form, which you'll see in many contexts
- Here's a typical empirical result for the polynomial order example...

Train and test curve

- Empirical result for the polynomial order example...

- Arguably single most important empirical phenomenon in learning!
  - Note that **training set error** goes to zero
  - But **test set error** finds a min then goes up and up
  - This is the point after which we're over-fitting

Overfitting in supervised learning

- We've seen that a snugly fit model can nonetheless be a poor predictor
- **Train/test** and **cross-validation** provide a means to check that a given class of model is useful
- But they are empirical and computationally intensive

Overfitting in supervised learning

- We've seen that a snugly fit model can nonetheless be a poor predictor

- **Train/test** and **cross-validation** provide a means to check that a given class of model is useful

- But they are empirical and computationally intensive:
  – Not usually practical for learning the *parameters* for a given class/ complexity of model
  – Better suited to checking a small set of models *after* parameter estimation

- Also, in some settings, a relatively complex model may make *sense*
- But the overfitting problem won't just go away, so it's important to methods to fit more complex models

Penalized likelihood

- The problem of overfitting is one of sticking too closely to the data, being overly reliant on the likelihood
- In regression, what happens is that we get large coefficients for inputs or functions of inputs
- E.g. For polynomial example:

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

- Natural idea: modify objective function to take account of size of weight vector...

Ridge regression

- Want to modify objective function to take account of size of weights
- One way is to add a term capturing the length of the weight vector:

$$
\begin{aligned}
J(\mathbf{w}) &= \|\mathbf{\Phi w} - \mathbf{Y}\|^2 + \lambda\|\mathbf{w}\|^2 \\
\mathbf{\Phi}_{ij} &= \phi_j(\mathbf{X}_i) \\
\mathbf{Y} &= [Y_1 \ldots Y_n]^T
\end{aligned}
$$

- This is called **ridge regression**
- Objective function is called a *penalized likelihood*, second term is an *"L2 penalty"*
- It ought to to discourage solutions with large weights

Ridge regression: learning

- Objective function:

$$\begin{aligned} J(\mathbf{w}) &= \|\mathbf{\Phi w} - \mathbf{Y}\|^2 + \lambda\|\mathbf{w}\|^2 \\ &= (\mathbf{\Phi w} - \mathbf{Y})^T(\mathbf{\Phi w} - \mathbf{Y}) + \lambda \mathbf{w}^T \mathbf{w} \end{aligned}$$

- Taking derivative wrt to ***w*** and setting to zero:

$$\mathbf{\Phi}^T(\mathbf{\Phi}\hat{\mathbf{w}} - \mathbf{Y}) - \lambda\hat{\mathbf{w}} = 0$$

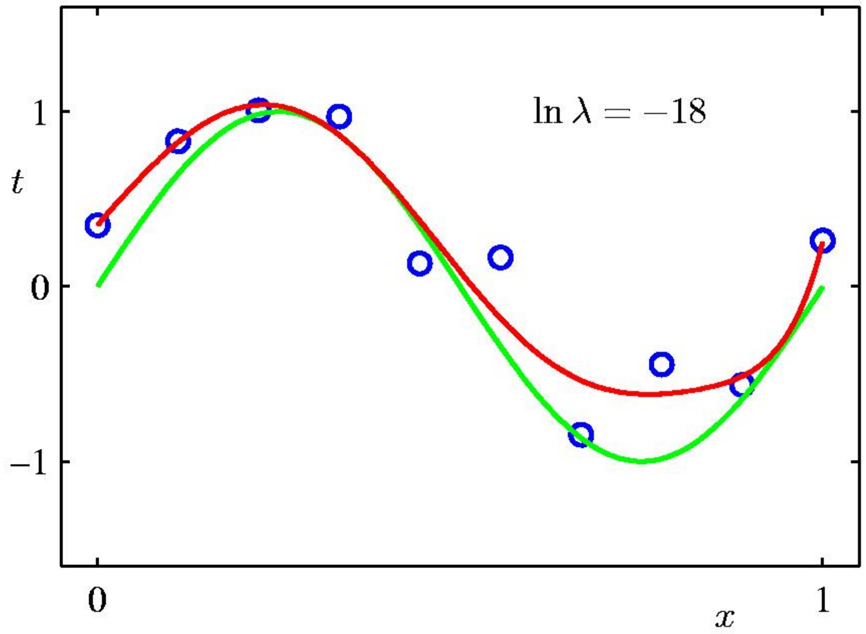
- Solving for ***w***:

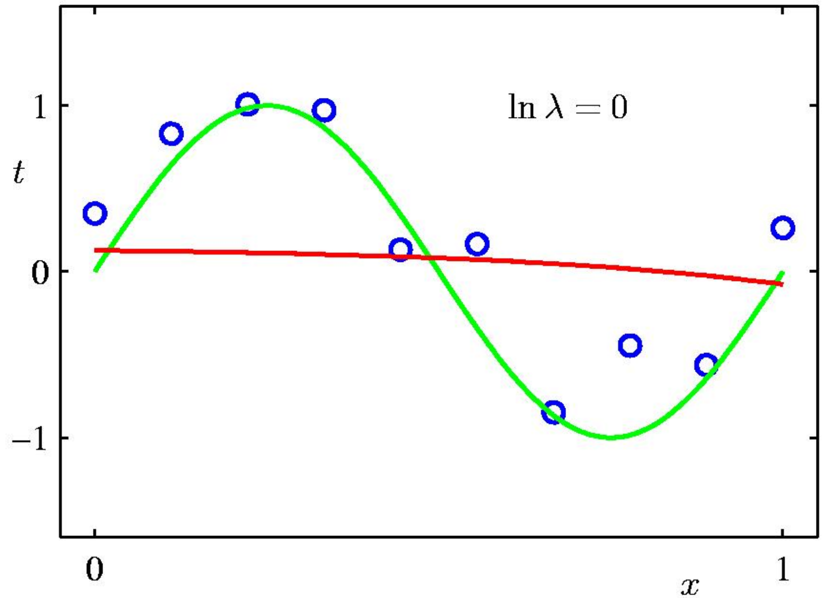
$$\hat{\mathbf{w}} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda \mathbf{I}_k)^{-1}\mathbf{\Phi}^T\mathbf{Y}$$

- Closed form solution
  - Can't use pseudo inverse trick
- Adding identity improves conditioning of matrix
  - (cf *Tikhonov regularization*)
- Let's try it for k=9

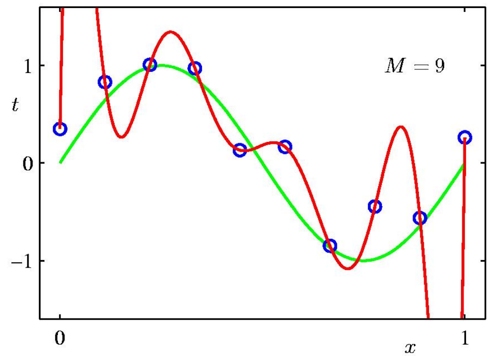
Ridge regression: learning

- Ridge (red dashed line)



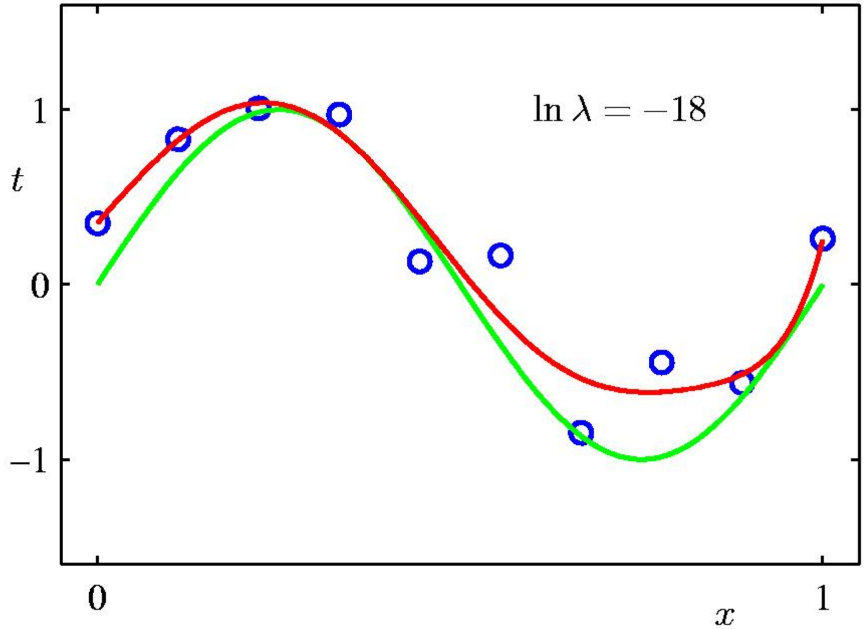


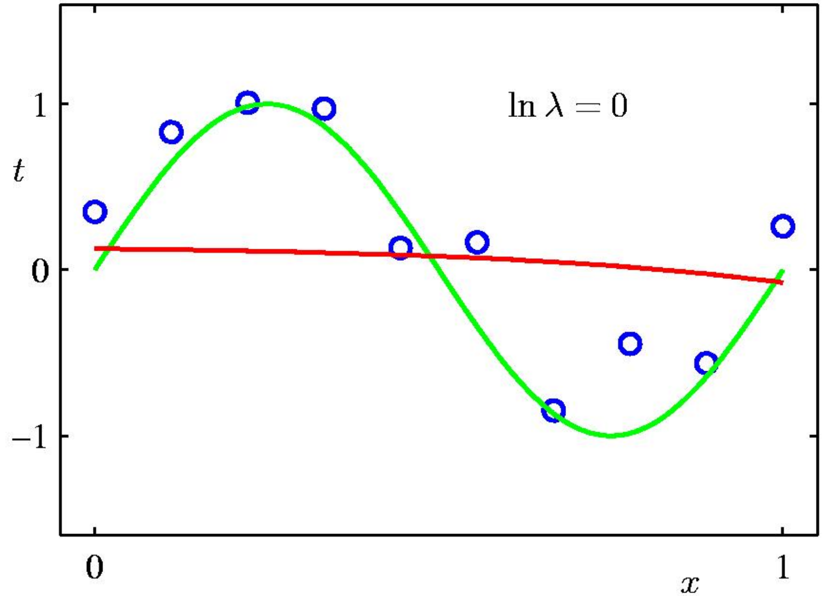
- Recall what the least squares/MLE for k=9 looked like...



Ridge regression: learning

- Ridge (red dashed line)

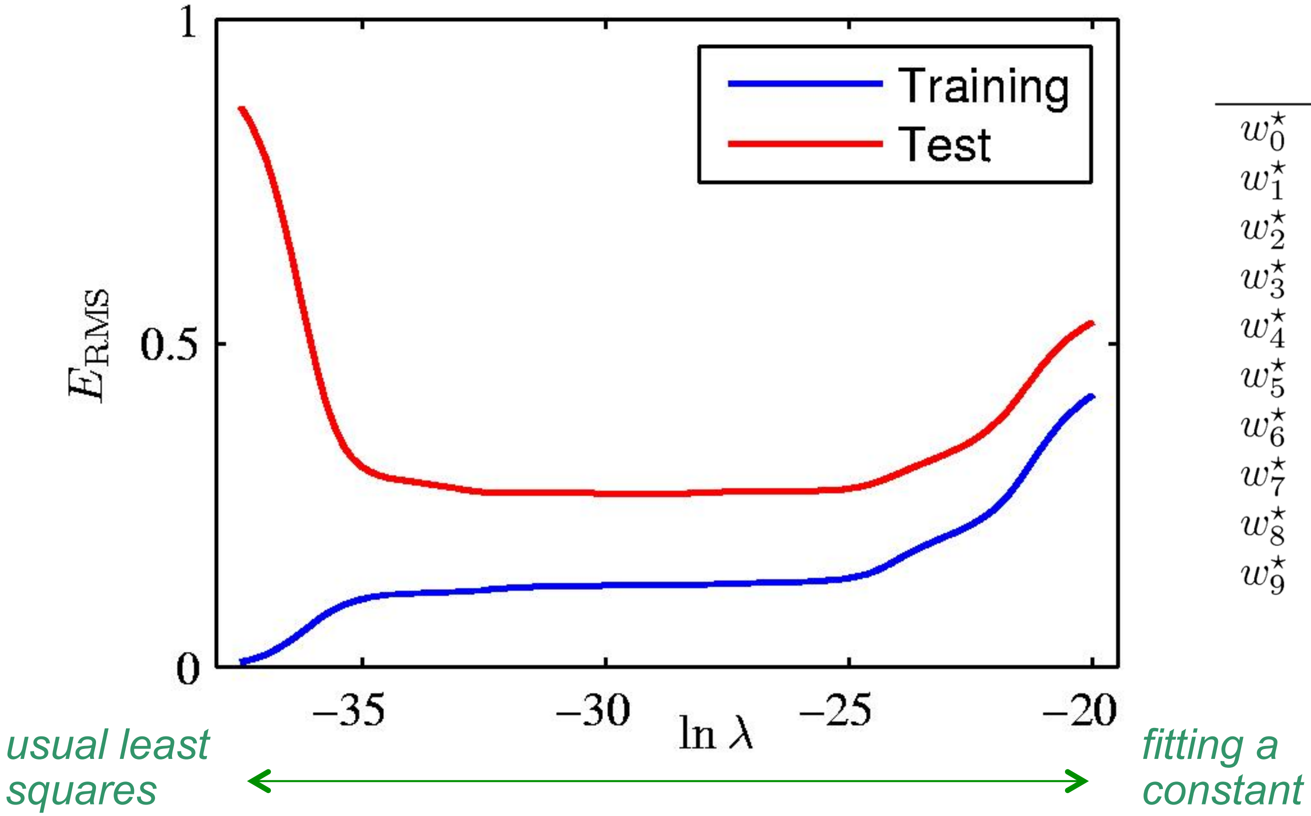




- **Ridge regression** is much better. The large values of the weight vector are kept under control and prediction is noticeably improved
- Ridge parameter can be learned by cross-validation

Ridge regression: learning

- Cross-validation to learn λ



	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Back to Bayes

- For the coins, a Bayesian approach was great
  - MAP estimate was nice alternative to the MLE
- What does Bayesian regression look like?

Bayesian regression

- Recall the likelihood model for regression:

$$p(Y \mid \mathbf{X}, \mathbf{w}, \sigma^2) \quad = \quad \mathcal{N}(Y \mid \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}), \sigma^2)$$

- Here, the weights are the unknown parameters of interest, so we should write down a **posterior distribution over the weights...**

Posterior over weights

- **Posterior distribution over weights**:

$$\begin{aligned} p(\mathbf{w} \mid \mathbf{Y}, \mathbf{X}) &\propto p(\mathbf{Y} \mid \mathbf{w}, \mathbf{X})\, p(\mathbf{w}|\mathbf{X}) \\ &= p(\mathbf{Y} \mid \mathbf{w}, \mathbf{X})\, p(\mathbf{w}) \end{aligned}$$

- This is just:

$$posterior \propto likelihood \times prior$$

- *p(**w**)* is a **prior**
- We'll use a **zero mean MVN**. This means that

(i) Weights are expected to be small (centred around zero)

(ii) Large deviations from zero are strongly discouraged (light tails)

Posterior over weights

- Prior on weights:

$$p(\mathbf{w}) \quad = \quad \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \sigma_0^2 \mathbf{I})$$

- This is a simple, one-parameter multi-variate density, the variance is a **hyper-parameter**

- Under the (conditionally) independent Normal model, the **posterior** is:

$$p(\mathbf{w} \mid Y_1 \ldots Y_n, \mathbf{X}_1 \ldots \mathbf{X}_n)$$

$$\propto \left[\prod_{i=1}^{n} \mathcal{N}(Y_i \mid \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i), \sigma^2) \right] \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \sigma_0^2 \mathbf{I})$$

MAP estimate of weights

- **Q: write down the log-posterior, and hence derive the MAP estimate of the weights**

MAP estimate of weights

- **Q: what is the MAP estimate of the weights?**

$$\log(p(\mathbf{w} \mid \mathbf{Y}, \mathbf{X}))$$

$$\propto n \log\left(\frac{1}{\sqrt{2\pi}\sqrt{\sigma^2}}\right) - \frac{1}{2}\sum_{i=1}^{n}\left(\frac{Y_i - \mathbf{w}^T \phi(\mathbf{X}_i)}{\sigma^2}\right)^2 + \log\left(\frac{1}{(2\pi)^{d/2}|\sigma_0^2 \mathbf{I}|^{1/2}}\right) - \frac{1}{2}\mathbf{w}^T(\sigma_0^{-2}\mathbf{I})\mathbf{w}$$

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmax}} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(Y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i))^2 - \frac{1}{2}\mathbf{w}^T(1/\sigma_0^2)\mathbf{I}\mathbf{w}$$

- Changing sign and multiplying through by σ^2:

$$\begin{aligned}
\hat{\mathbf{w}}_{MAP} &= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n}(Y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{X}_i))^2 + \frac{\sigma^2}{\sigma_0^2}\mathbf{w}^T\mathbf{w} \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \|\boldsymbol{\Phi}\mathbf{w} - \mathbf{Y}\|^2 + \lambda\|\mathbf{w}\|^2
\end{aligned}$$

MAP estimate of weights

$$\begin{aligned}\hat{\mathbf{w}}_{MAP} &= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (Y_i - \mathbf{w}^T \phi(\mathbf{X}_i))^2 + \frac{\sigma^2}{\sigma_0^2} \mathbf{w}^T \mathbf{w} \\ &= \arg\min_{\mathbf{w}} \|\mathbf{\Phi w} - \mathbf{Y}\|^2 + \lambda \|\mathbf{w}\|^2\end{aligned}$$

- But this is simply **ridge regression**!
  - Penalty λ is ratio of residual variance to prior variance
- Unsurprising: prior was Normal, the quadratic term in the exponent corresponds to the L2 penalty in ridge regression

- Thus, we get:

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{\Phi}^T \mathbf{\Phi} + \lambda \mathbf{I}_k)^{-1} \mathbf{\Phi}^T \mathbf{Y}$$

Regression

- Simple, closed form solution for linear-in-parameters problems
- Complex models give power to fit interesting functions, but run the risk of overfitting

- Penalized likelihood methods like Ridge regression, or Bayesian approaches allow us to fit complex models while ameliorating over-fitting
- Train/test, cross validation are valid ways to check how well we're doing