Lecture 5: Regularisation, Multivariate Models, and Nonlinear Problems

Attendance code: K8PE3UZF

Iain Styles

25 October 2019

Learning Outcomes

By the end of the this lecture you should be able to

- Understand the effect of regularisation on regression problems
- Be able to formulate and solve linear regression problems with multiple independent and dependent variables
- Appreciate how non-linear problems are different, and how they can be approached.

Recap

- Uniform prior on model parameters $\mapsto$ Least-sqaures loss
- Gaussian prior on model parmaeters $\mapsto$ least squares loss with L_2 penalty term
- Encourages model parameters to not grow too large
- Helps to reduce overfitting
- How do we apply it?

Regularised Loss Functions

- Generalised regularised loss function

$$\mathcal{L}(\mathbf{w}) = \mathcal{L}_{\text{err}}(\mathbf{w}) + \lambda R(\mathbf{w}) \tag{1}$$

- Model-data mismatch plus "penalty" term
- For the specific case of LSE + Ł$_2$ penalty (Gaussian prior)

$$\mathcal{L}(\mathbf{w}) = (\mathbf{y} - \mathbf{\Phi w})^{\mathrm{T}} (\mathbf{y} - \mathbf{\Phi w}) + \lambda \mathbf{w}^{\mathrm{T}} \mathbf{w} \tag{2}$$

- Minimising loss requires both terms to be minimised
- λ controls the width of the Gaussian prior and hence the balance between model fitting and parameter shrinkage.

Solving Regularised Least Squares

- One reason L_2 is common is its closed-form analytic solution
- Differentiate loss function with respect to $\mathbf{w}$ and set to zero to minimise

$$\begin{aligned} \mathcal{L}(\mathbf{w}) = & \quad (\mathbf{y} - \mathbf{\Phi w})^{\mathrm{T}} (\mathbf{y} - \mathbf{\Phi w}) + \lambda \mathbf{w}^{\mathrm{T}} \mathbf{w} \\ \frac{\partial \mathcal{L}_{\mathrm{LSE}}(\mathbf{w})}{\partial \mathbf{w}} = & \quad -2\mathbf{\Phi}^{\mathrm{T}} (\mathbf{y} - \mathbf{\Phi w}) + 2\lambda \mathbf{w} \end{aligned}$$

- Set to zero to minimise

$$\mathbf{\Phi}^{\mathrm{T}} \mathbf{y} - \left(\mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi} - \lambda \mathbf{I} \right) \mathbf{w}^* = 0$$

- Modified normal equations, can be solved in the same way

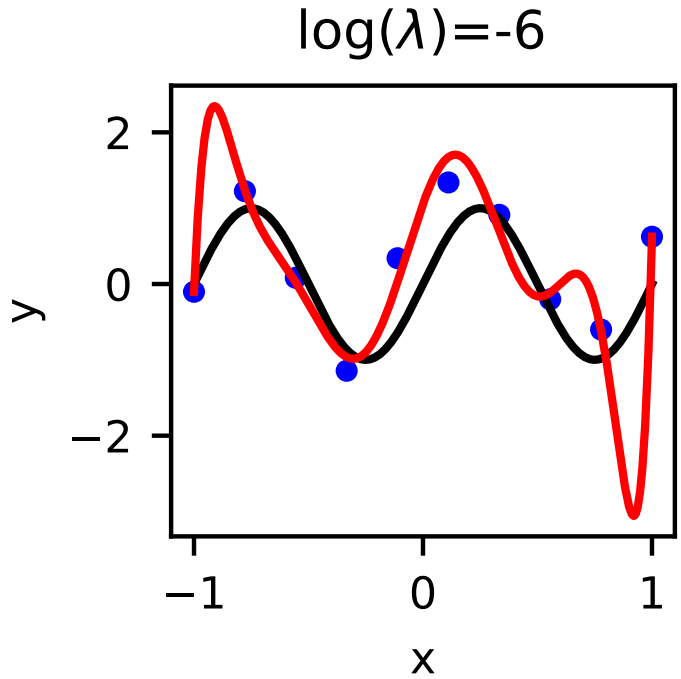
Terminology

Known by a number of names:

- *Ridge regression*.
- *L_2 regularisation*, because $\sum_i w_i^2$ is the L_2 norm of $\mathbf{w}$, written as $||\mathbf{w}||_2^2$.
- *Weight decay*, because it pushes weights towards zero.
- *Tikhonov regularisation*, of which it is a special case. Tikhonov regularisation uses $R(\mathbf{p}) = ||\mathbf{\Gamma w}||$. Here, we have $\mathbf{\Gamma} = \lambda \mathbf{I}$.

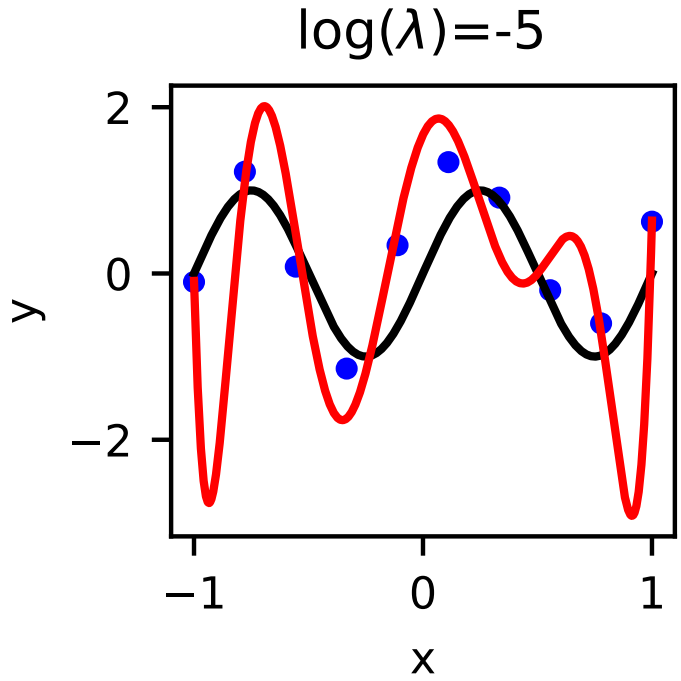
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



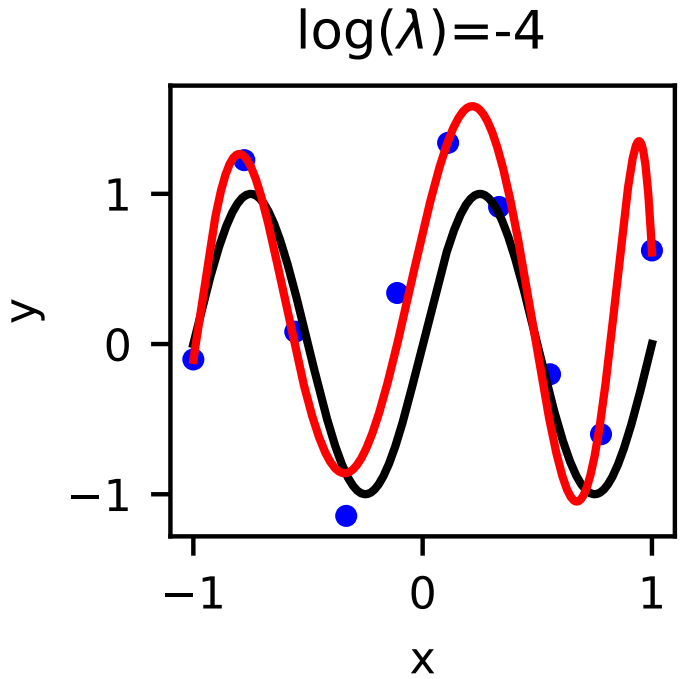
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



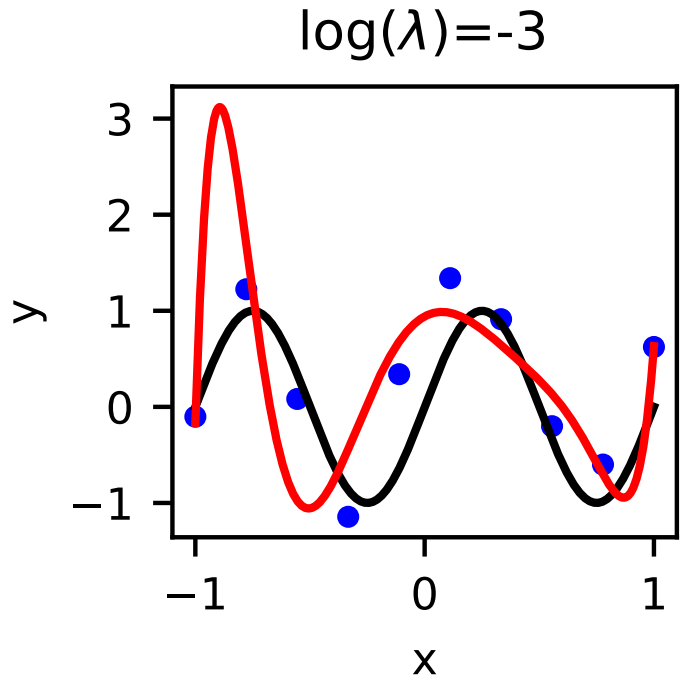
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



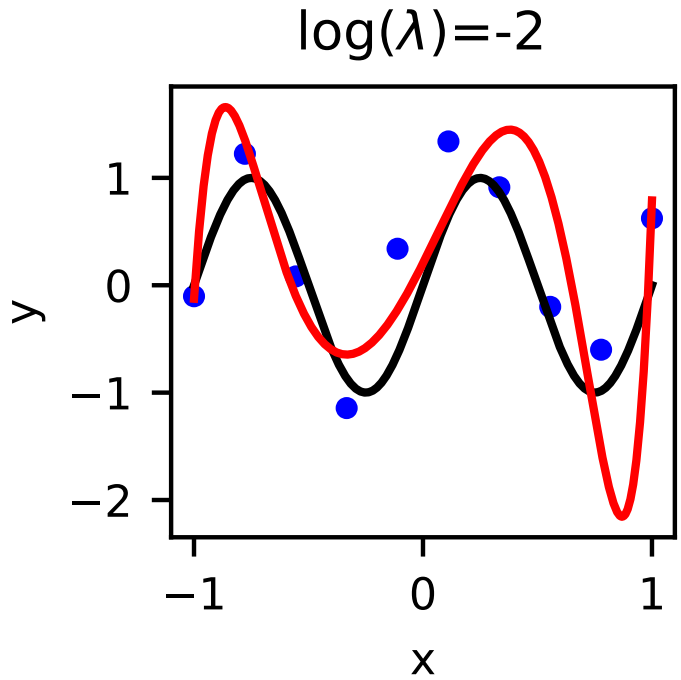
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



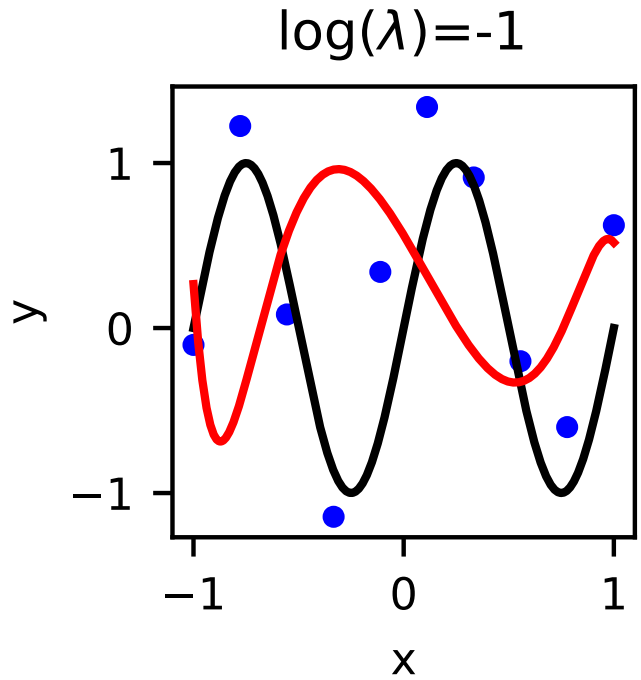
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



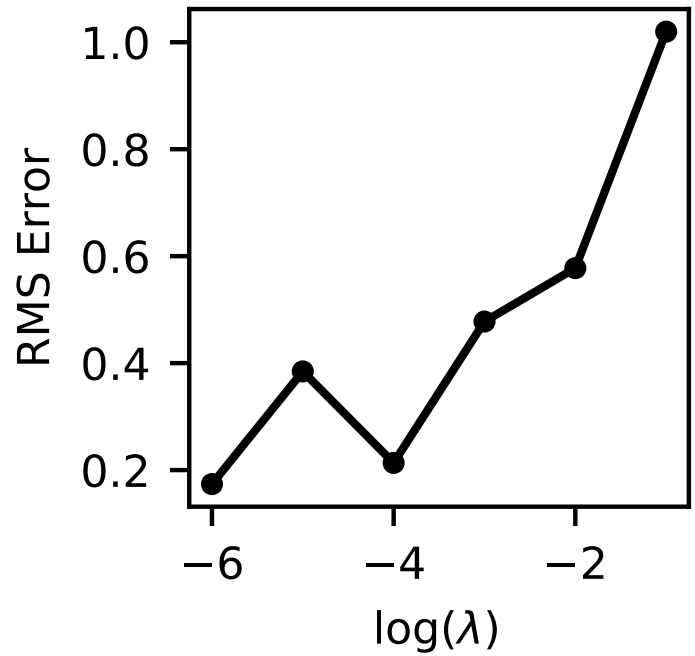
Regularisation in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



Effect of Regularisation

- Regularisation increases training error but improves generalisation
- Penalty term reduce model weights



$\log_{10} \lambda$	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
-6	1.06	8.31	-17.92	-76.20	76.16	250.58	-112.92	-350.94	53.88	168.61
-5	1.67	5.77	-41.44	-29.12	212.84	31.81	-356.24	1.30	183.44	-9.40
-4	0.74	6.51	-5.21	-33.75	1.46	33.96	20.81	13.89	-17.55	-20.25
-3	0.94	1.29	-9.30	5.42	15.29	-21.02	5.50	-4.36	-12.20	19.08
-2	0.20	4.26	2.26	-10.00	-5.63	-4.88	-0.83	2.60	4.33	8.47
-1	0.56	-2.12	-1.35	4.00	-0.63	1.44	0.50	-0.84	1.30	-2.35

Limitations

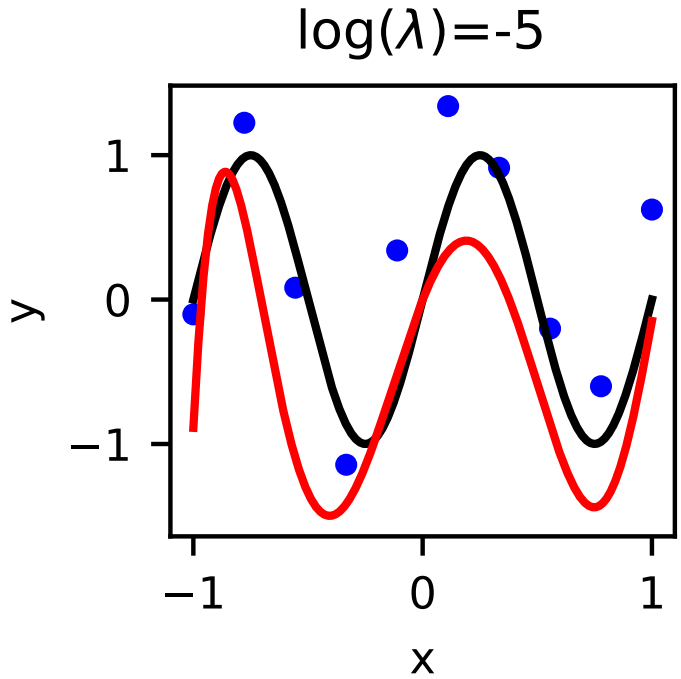
- L_2 regularisation tends to "smooth" the fit
- Popular for image denoising
- But does not work well for data that really does have "fast" fluctuations.
- Other choices can overcome this
- Most general form: $R(\mathbf{w}) = \sum_i |w_i|^p$
- $p = 1$ is very common - L_1 or *lasso* regularisation

L_1 Regularisation

- Also known as the Lasso methods
- $R(\mathbf{w}) = \sum_i |w_i|$
- Tends to promote *sparsity* in model parameters
- No closed form solution
- Turn to `scikit-learn` for implementation

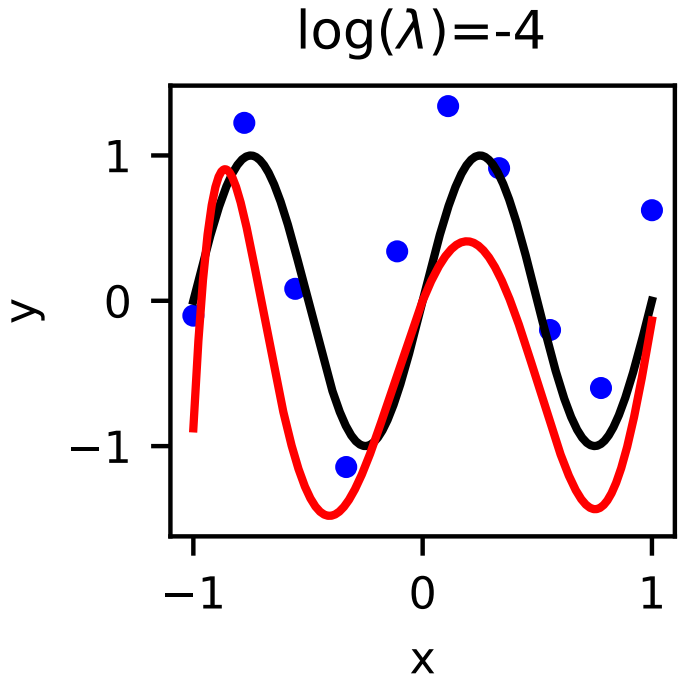
Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



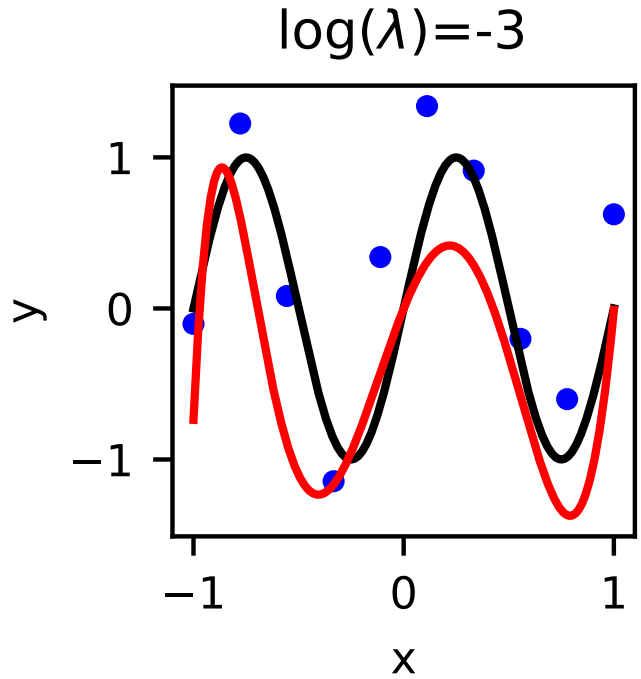
Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



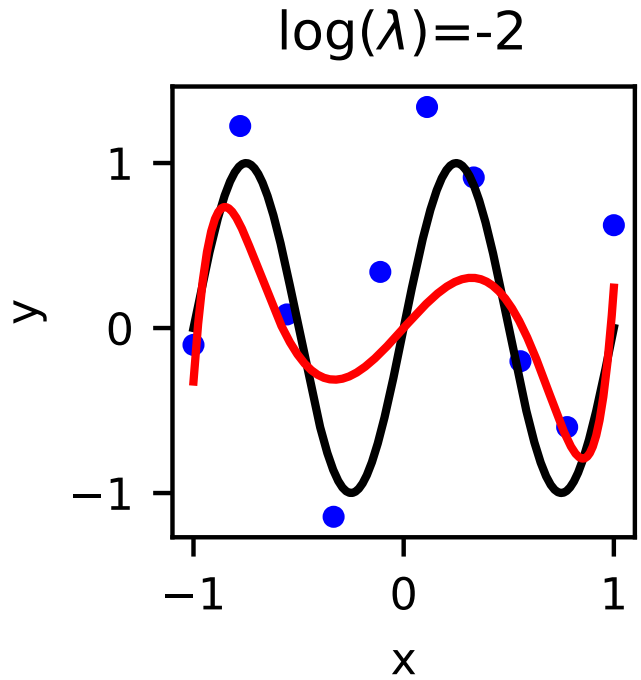
Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



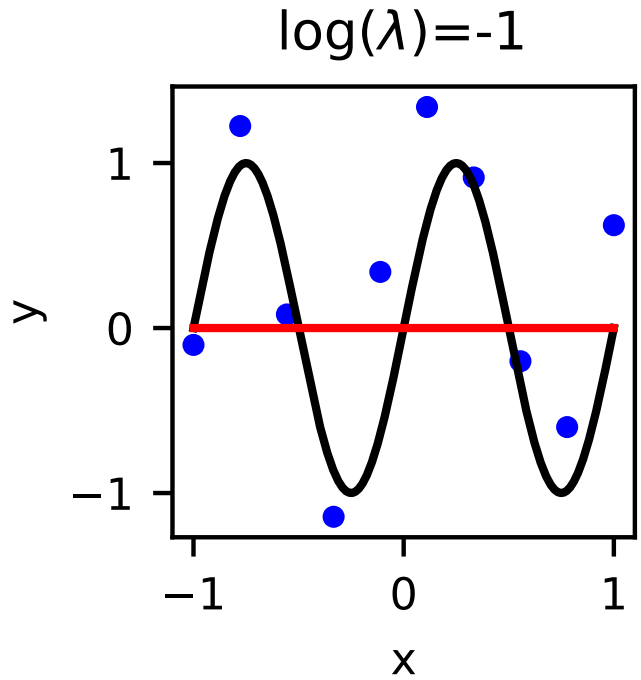
Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



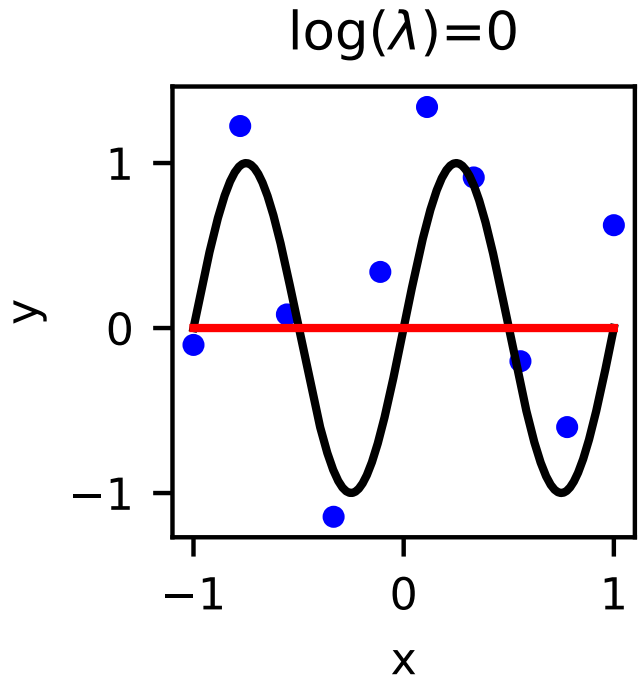
Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



Lasso in Practice

- L_2 regression used to prevent overfitting
- Prevents model weight growing large to fit noise.
- Large values of λ "smooth" fluctuations



L_1-regularised model parameters

$\log_{10} \lambda$	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
-5	0.00	3.59	-0.00	-15.72	0.00	11.78	0.00	1.96	0.00	-1.60
-4	0.00	3.54	0.00	-15.27	0.00	11.05	0.00	1.93	0.00	-1.24
-3	0.00	3.10	-0.00	-12.02	-0.00	5.44	-0.00	3.49	-0.00	0.00
-2	0.00	0.95	-0.00	-3.76	-0.00	-0.00	-0.00	0.00	-0.00	2.74
-1	0.00	-0.05	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00
0	0.00	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00

- Progressice sparsification of model parameters

Regularisation in Practice

- The correct regulariser and choice of λ is very problem dependent
- Smooth or sparse solution?
- Rigorous cross-validation often needed to selection λ
- Enables complex models to be used and then controlled

Regularisation in Practice

- The correct regulariser and choice of λ is very problem dependent
- Smooth or sparse solution?
- Rigorous cross-validation often needed to selection λ
- Enables complex models to be used and then controlled
- And now a change of topic...

Multivariate Regression

- One independent variable x, one dependent variable y
- How to deal with multiple variables of each type?
- Univariate methods generalise quite straightforwardly
- Consider two independent variables, and a model that contains a constant term and linear terms in the two variables:

$$y = w_0 + w_1 x_1 + w_2 x_2$$

Multivariate Basis Matrix

- Same form as polynomial expansion
- Basis matrix has one row per sample, one column per feature

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} \\ 1 & x_{1,2} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{1,N} & x_{2,N} \end{pmatrix}$$

Multivariate Basis Matrix

- Same form as polynomial expansion
- Basis matrix has one row per sample, one column per feature

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} \\ 1 & x_{1,2} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{1,N} & x_{2,N} \end{pmatrix}$$

- A more complex quadratic model is similar

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$

has basis matrix

$$\mathbf{\Phi} = \begin{pmatrix} 1 & x_{1,1} & x_{2,1} & x_{1,1}^2 & x_{2,1}^2 & x_{1,1}x_{2,1} \\ 1 & x_{1,2} & x_{2,2} & x_{1,2}^2 & x_{2,2}^2 & x_{1,2}x_{2,2} \\ \vdots & \vdots & \vdots & & & \\ 1 & x_{1,N} & x_{2,N} & x_{1,N}^2 & x_{2,N}^2 & x_{1,N}x_{2,N} \end{pmatrix}$$

Multiple Dependent Variables

- Assumes same set of features/basis for each output
- Approach 1: treat all outputs separately:

$$
\begin{aligned}
y_1(x_1,\ldots,x_K) &= \sum_j w_{1,j}\phi_j(x_1,\ldots,x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_1 = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_1 \\
y_2(x_1,\ldots,x_K) &= \sum_j w_{2,j}\phi_j(x_1,\ldots,x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_2 = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_2 \\
&\vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\
y_L(x_1,\ldots,x_K) &= \sum_j w_{L,j}\phi_j(x_1,\ldots,x_K) \quad \rightarrow \quad \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\mathbf{w}_L = \mathbf{\Phi}^{\mathrm{T}}\mathbf{y}_L
\end{aligned}
$$

where $\mathbf{y}_j = [y_{j,1}, y_{j,2}, \ldots, y_{j,N}]^{\mathrm{T}}$ and $\mathbf{w}_i = [w_{i,1}, w_{i,2}, \ldots w_{i,M}]$.

Multiple Dependent Variables

- Approach 2: "joint" optimisation over all of the dependent variables in a single step

$$\begin{pmatrix} y_{1,1} & y_{2,1} & \cdots & y_{K,1} \\ y_{1,2} & y_{2,2} & \cdots & y_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1,N} & y_{2,N} & \cdots & y_{K,N} \end{pmatrix} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_M(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_M(x_N) \end{pmatrix} \times \begin{pmatrix} w_{1,1} & w_{2,1} & \cdots & w_{K,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,M} & w_{2,M} & \cdots & w_{K,M} \end{pmatrix}$$

$$\rightarrow \mathbf{Y} = \mathbf{\Phi W}$$

- Joint solution via normal equations $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi W} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{Y}$

Non-linear Regression

- Non-examinable
- How to deal with nonlinear models, eg $y(x) = a \sin bx$, where we need to find a and b.
- Modelling function is not linear in unknowns
- Cannot minimise loss directly
- Instead, locally reduce the loss
- Do this iteratively to find global minimum

Locally linearise the loss

- Approximating loss as a straight line over a small region using *Taylor's Theorem*:

$$\begin{aligned} f(x_i, \mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) &\approx f(x_i, \mathbf{w}) + \sum_{j=1}^{m} \frac{\partial f(x_i)}{\partial w_j} \Delta w_j \\ &= f(x_i, \mathbf{w}) + \sum_{j=1}^{m} J_{ij} \delta w_j \end{aligned}$$

  where matrix $\mathbf{J}$ has components $J_{ij} = \frac{\partial f(x_i)}{\partial w_j}$

- The residual becomes

$$\mathbf{r}(\mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) = \mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w}$$

  where $f_i(\mathbf{w}) = f(x_i, \mathbf{w})$. The loss is therefore

$$\mathcal{L}(\mathbf{w} + \boldsymbol{\Delta}\mathbf{w}) = (\mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w})^{\mathrm{T}} (\mathbf{y} - \mathbf{f}(\mathbf{w}) - \mathbf{J}\boldsymbol{\Delta}\mathbf{w}) .$$

  By minimising this quantity, we find the *change* in $\mathbf{w}$ that gives us the largest reduction in the error.

Locally minimise the loss

- We differentiate as before and set to zero:

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)\boldsymbol{\Delta}\mathbf{w}^{*} = \mathbf{J}^{\mathrm{T}}\left(\mathbf{y} - \mathbf{f}(\mathbf{w})\right)$$

  which we now how to solve

- It is common practice to make a small modification to this:

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\,\mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J})\right)\boldsymbol{\Delta}\mathbf{w}^{*} = \mathbf{J}^{\mathrm{T}}\left(\mathbf{y} - \mathbf{f}(\mathbf{w})\right)$$

- Regularisation that penalises large updates to the parameter vector along directions with steep gradients, encouraging movement along shallow gradients and thus a faster convergence.

The Levenberg-Marquadt Algorithm

Data: Data $\mathbf{x}, \mathbf{y}$; Initial parameter guess $\mathbf{w}_0$; Function $f(x, \mathbf{w})$; Jacobian $Jac(x, \mathbf{w})$; initial values of λ, ν.
Result: $\mathbf{w}, e(\mathbf{w})$, parameters that minimmise the least squares error, the error.

```
% Evaluate the error at the initial parameter value
w ← w_0;
e ← (y − f(x, w))^T (y − f(x, w));
% Iterate until convergence
Converged ← False;
while Converged == False do
    % Calculate the Jacobian
    J ← ∂f/∂w;
    % Calculate update for different values of λ until we improve the error
    LambdaSet ← false;
    while LambdaSet==false do
        % Calculate Δw for λ = λ_0 and λ = λ_0/ν
        Δ_1 = (J^T J + λ diag(J^T J)) \ J^T (y − f(x, w));
        Δ_2 = (J^T J + (λ/ν) diag(J^T J)) \ J^T (y − f(x, w));
        % Calculate the error at the new parameter values
        e_1 ← (y − f(x, w + Δ_1))^T (y − f(x, w));
        e_2 ← (y − f(x, w + Δ_2))^T (y − f(x, w));
        if e_1 < e then
            % Set new values of w and e, storing old value as e'
            w ← w + Δ_1;
            e' ← e;
            e ← e_1;
            % Current λ is OK
            LambdaSet ← true;
        else if e_2 < e then
            % Set new values of w and e, storing old value as e'
            w ← w + Δ_2;
            e' ← e;
            e ← e_2;
            λ ← λ/ν;
            % Current λ is OK
            LambdaSet ← true;
        else
            λ ← λ × ν;
    end
    if e'/e < TerminationCriterion then
        Converged ← true
    end
end
```

Algorithm 1: The Levenberg

- Effective and powerful method for non-linear problems
- But only if loss is convex
- Will find local minima and get stuck
- In those cases, use evolutionary/genetic algorithms

Summary

- Two different regularisation methods
- Multivariate regression problems
- Nonlinear problems (not examinable)
- The end of regression
- Next: classification