Imperial College London

Supervised Learning: linear regresssion

Machine Learning

QSRI summer school – July 2022

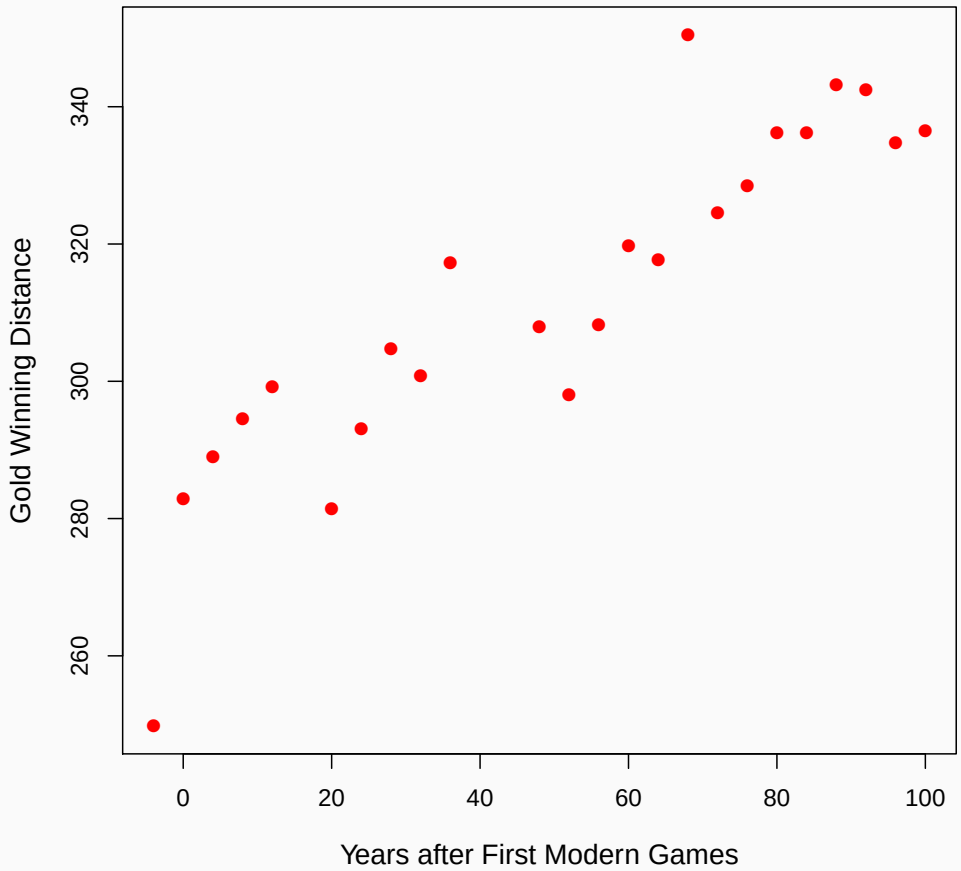
Sarah Filippi

Department of Mathematics
Imperial College London

An Example of Regression

Can you predict the gold winning distance in 2012,
i.e. 112 years after the first modern game?

Vote by going to https://www.menti.com and using the code **7268 1021**.



Supervised learning

A training dataset consists of $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$

- input variables $x^{(i)} \in \mathcal{X}$, $i = 1 \dots N$
- and response variables $y^{(i)} \in \mathcal{Y}$, $i = 1 \dots N$.

Types of supervised learning:

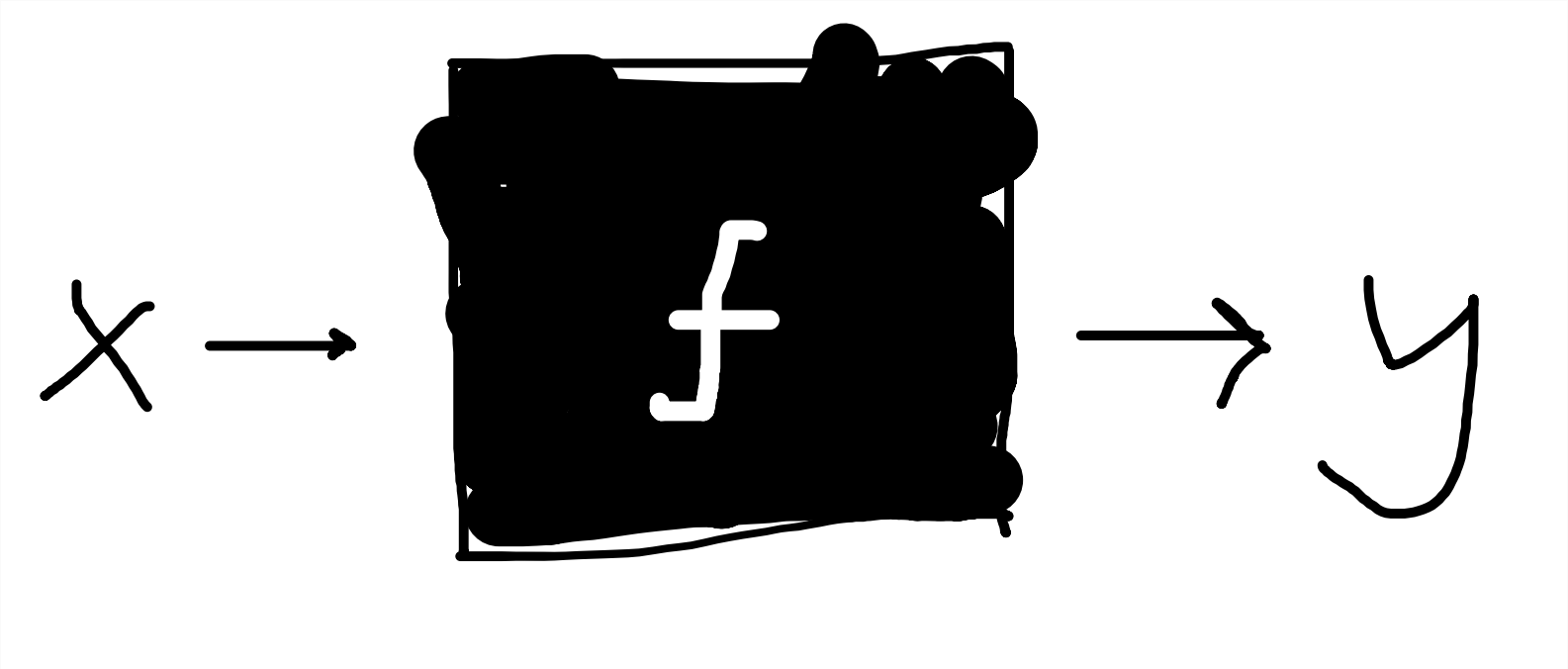
- Regression: a numerical value is observed, e.g. $\mathcal{Y} = \mathbb{R}$.
- Classification: discrete responses, e.g. $\mathcal{Y} = \{+1, -1\}$ or $\{1, ..., K\}$.

The goal is to accurately predict the response y on new observations of x, i.e. to learn a function

$$f : \mathcal{X} \mapsto \mathcal{Y}$$

such that $f(x)$ will be close to the true response y.

Supervised learning



The function (algorithm, black box, decision rule, classifier, probability distribution) $f : \mathcal{X} \mapsto \mathcal{Y}$ encapsulates our assumption regarding the underlying mechanism that generates the observed data.

Lectures' Objectives

- Introduce **linear models** as an example of supervised learning approach
- Introduce learning based on the concept of a **loss function**
- Consider the effect that **model complexity** has on **predictive ability**
- Introduce **cross validation** as a means of determining performance and choosing between models.
- Introduce **regularisation** techniques to prevent overfitting.

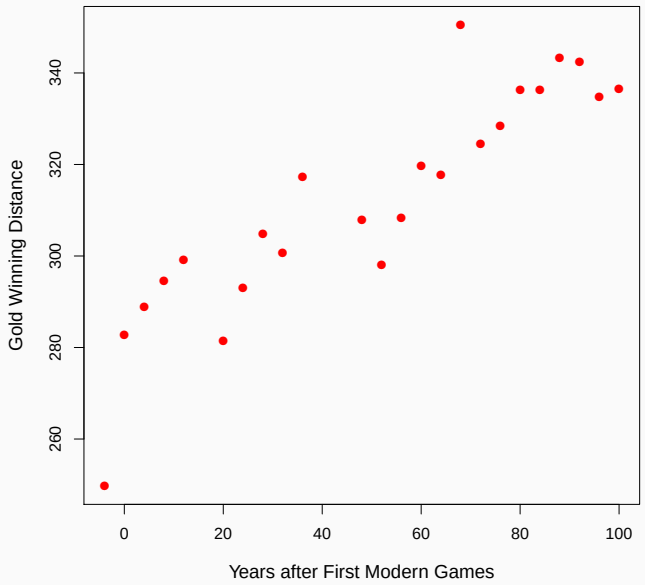
Linear models and loss functions

Model complexity, generalisation and cross-validation.

Variable selection and regularisation

Linear models and loss functions

An Example Dataset



In order to analyse this data using supervised learning, we have to decide upon the choice of functions (hypotheses) that we think might be appropriate.

We want to make predictions about future or unknown responses given new values of the attributes.

How do we learn the relationship from a finite set of observations?

Linear Models

Let assume that $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$.

In the case of **linear regression**, we consider functions $f : \mathcal{X} \mapsto \mathcal{Y}$ that are linear. A simple linear relationship is

$$f(x) = \alpha + \beta x \tag{1}$$

Loss Functions

Now that we have our data and have decided upon a model, how do we evaluate the model?

Given an input x and an output y,

$$\text{does} \quad f(x) \approx y \quad ?$$

In machine learning, we evaluate the models using a **loss function**.

A standard choice of loss function in regression is the square error:

$$L(y, f(x)) = (y - f(x))^2 \, .$$

Loss is bad, you want to avoid loss, so smaller loss is better! (Some losses are always positive, others can be positive or negative.)

Loss Functions and model training

As we will discuss along this course, the loss function can be used for multiple purposes. One of them is to **train a model**, i.e. learn the parameters of the model.

Given a training data, $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, we learn the model parameters by **minimizing the average loss on the training dataset**, i.e.

$$\frac{1}{N}\sum_{i=1}^{N} L\left[y^{(i)}, f(x^{(i)})\right] \tag{2}$$

When using the square error as a loss function, we obtain the sample mean squared error (MSE):

$$\frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)})\right)^2 \tag{3}$$

(See demo `lossLinearModel.R`.)

Learning parameters of a linear model

In order to learn the parameters of the linear model (α and β), we therefore *minimise* this loss function:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - f(x^{(i)})\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \alpha - \beta x^{(i)}\right)^2 \qquad (4)$$

Learning as optimization:
In this case, we can obtain an analytic solution to this minimisation problem,

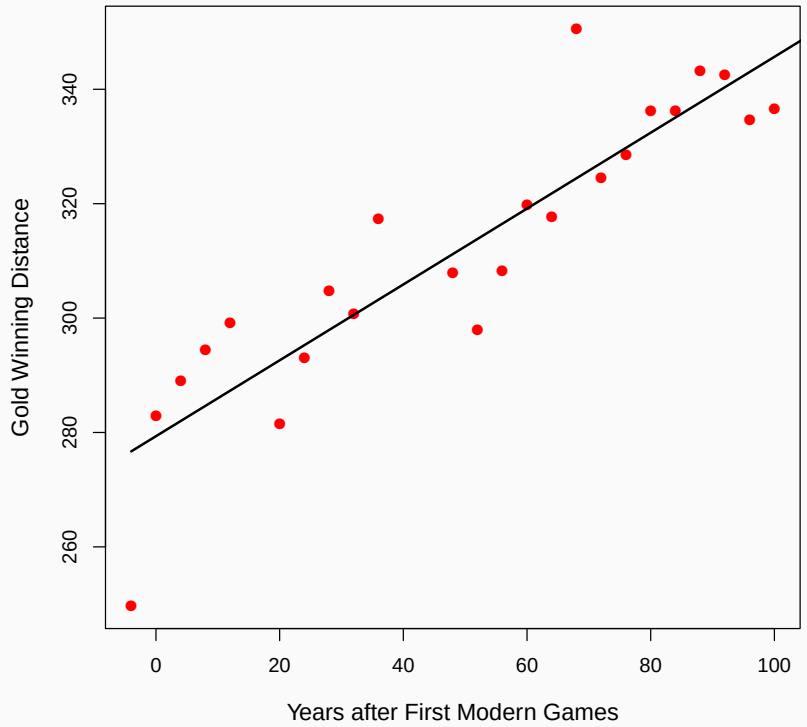
1. take the partial derivatives of the MSE function and set them to zero to find the stationary point;
2. check that this is indeed a minimum! This turns out to be the case here.

Note that for many other problems we must resort to numerical optimisation such as gradient descent, Newton's methods...

Making Predictions

The least squares estimate for the long jump data is

$$\begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix} = \begin{pmatrix} 279.3475 \\ 0.6633 \end{pmatrix}.$$



We can then use the parameter values we have learned to make predictions at a new value x^*

$$\hat{y^*} = \hat{\alpha} + \hat{\beta}x^* \tag{5}$$

Prediction on unseen data

Let's use the model to predict the winning distance in the 2012 Olympics since this data point was not included in our analysis:

$$\hat{y}_{2012} = \hat{\alpha} + \hat{\beta} \times 112 = 353.6 \tag{6}$$

In act, Greg Rutherfod's winning jump of just 327 inches was the shortest distance to win the men's long jump competition since the 1972 Summer Olympics!

It seems our model is too optimistic when making future predictions, which will only become longer and longer...

Using a linear function of time to predict future performances, is therefore probably not the best idea.

Linear model in more than one dimension

So far we considered that we only had one input variable x. Let now assume that the input $x = (x_1, \dots x_p) \in \mathbb{R}^p$. The linear model can be written

$$f(x) = \beta_0 + \sum_{m=1}^{p} \beta_m x_m \, . \tag{7}$$

We can represent the function f as an inner product between the parameter vector and the "feature vector":

$$f(x) = \underbrace{\begin{bmatrix} 1 & x_1 & x_2 \cdots & x_p \end{bmatrix}}_{\text{feature vector}} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

Note that we have added the constant 1 to the feature vector.

Loss Functions

Given a training data, $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, the mean square error

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \beta_0 - \sum_{m=1}^{p}\beta_m x_m^{(i)}\right)^2 \tag{8}$$

can be written more concisely in a matrix form

$$\text{MSE} = \frac{1}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{9}$$

where

$$\boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ \vdots & \vdots & & & \\ 1 & x_1^{(N)} & x_2^{(N)} & \dots & x_p^{(N)} \end{pmatrix},$$

The matrix $\mathbf{X}$ is known as the design matrix.

The vector of parameters that *minimise* this loss function is

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{10}$$

Fitting linear model in R

In `R` we use the function `lm` to fit a linear model.

Consider a data frame called `dataf` containing three variables `y`, `x1` and `x2` and suppose we want to predict `y` as a function of `x1` and `x2`.

We can learn the parameters of the linear model

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ .$$

using the function

```
fit <- lm(y ~ x1 + x2, data=dataf)
```

You can then use the commands `summary`, `coef`, `predict` to extract the model parameters or make prediction.

Summary

We have seen how to use the loss function to learn the parameters of a simple regression model.

Minimising the loss function to train a model is a key concept in Machine Learning.

For linear model, this minimisation problem can be solved analytically.

Next we will discuss how to use the loss function to compare models.

Model complexity, generalisation and cross-validation.

Learning the model and model complexity

We have seen how we can learn the *parameters* of the model based on the concept of a loss function.

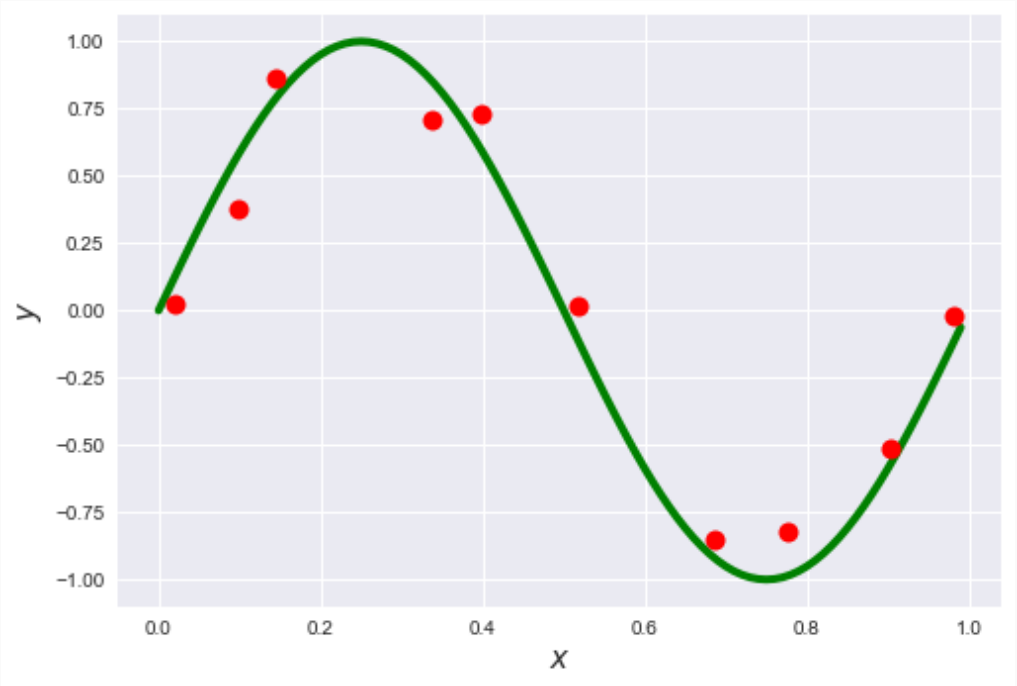
But if we have multiple models, how do we choose which *model* is the best description of the data?

Clearly if we increase the complexity of the model, we will be able to continually lower the loss function on the training data.

Does a lower loss on the training data imply a better model?

Underfitting vs overfitting

Consider data points generated by noisy observations of the function $\sin(2\pi x)$.



We consider using polynomial (of degree M) regressors of the form:

$$f(x) = \sum_{m=0}^{M} \beta_m x^m$$

Polyniomial regressor

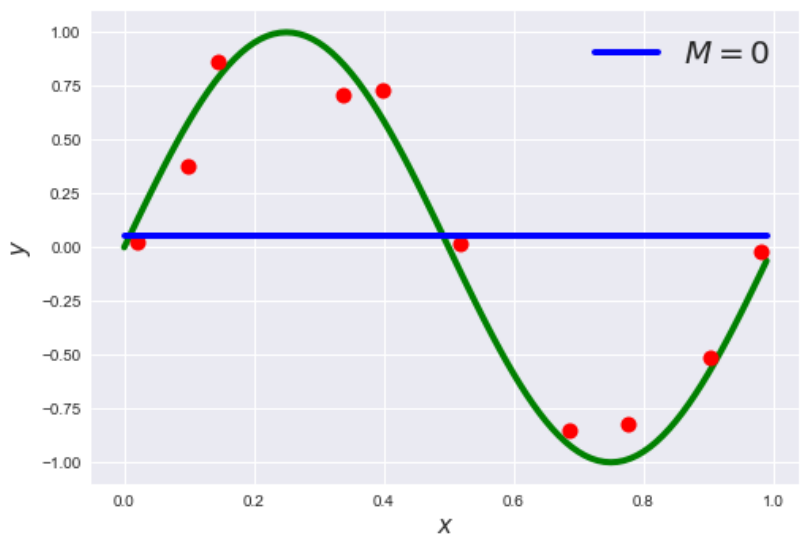
Note that a polynomial (of degree M) can be written on the form

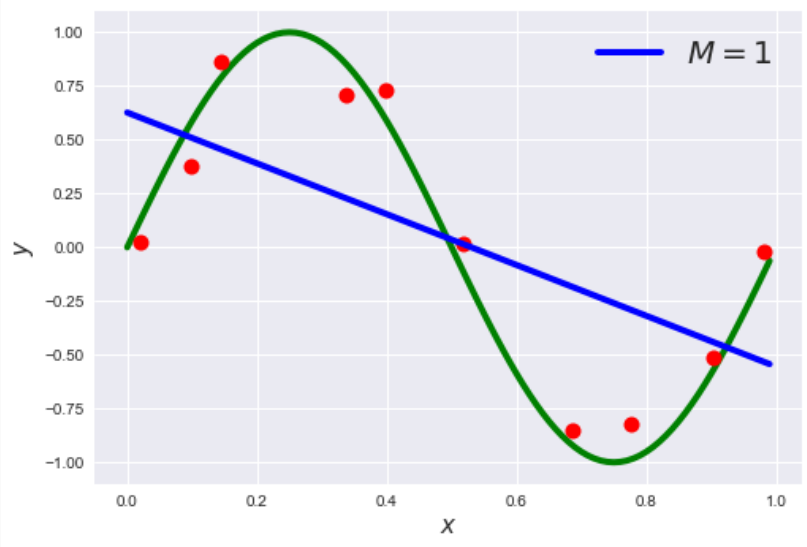
$$f(x) = \sum_{m=0}^{M} \beta_m x^m = \underbrace{\begin{bmatrix} 1 & x & x^2 \cdots & x^M \end{bmatrix}}_{\text{feature vector}} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_M \end{bmatrix}$$

and the parameters $(\beta_0, \beta_1, \ldots \beta_M)$ can be learnt by minimising the mean square error as for a linear model.

Underfitting vs overfitting

With $M = 0$ and $M = 1$ the regressor models are very simple (constant value and straight line respectively), and don't capture the data well:

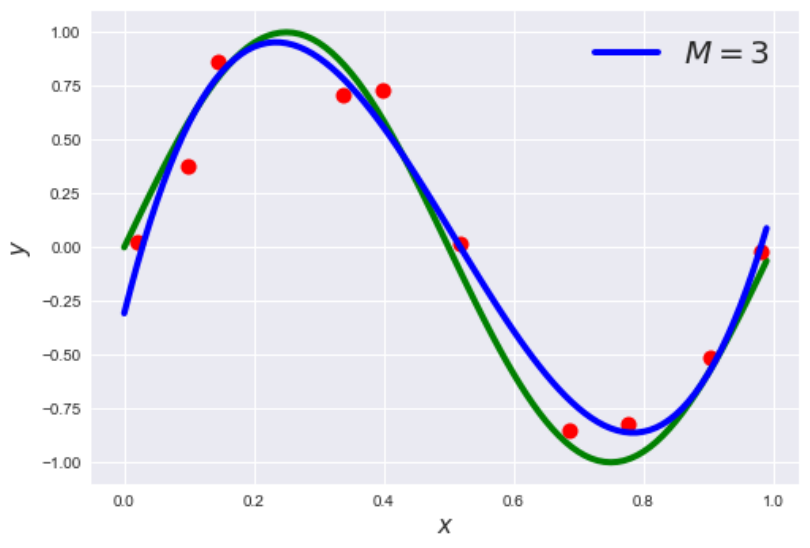


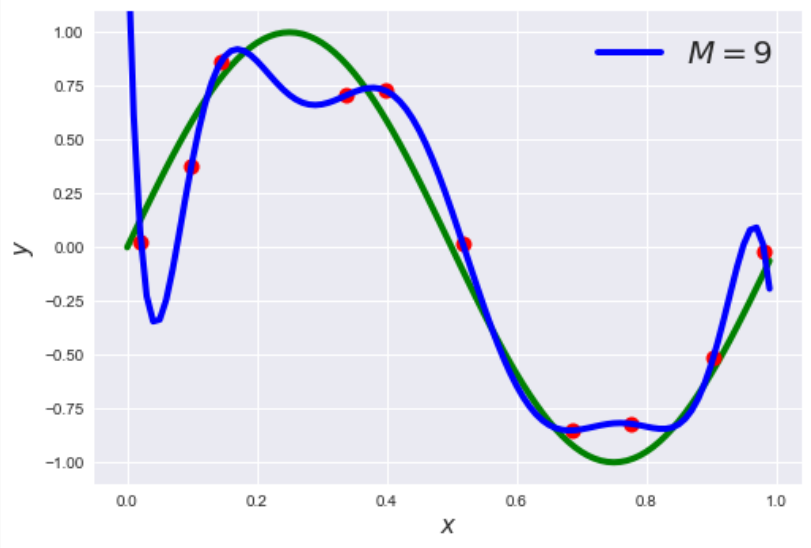


These models underfit the data.

Underfitting vs overfitting

Setting $M = 3$ gives a good fit to the data, but $M = 9$ leads to overfitting:

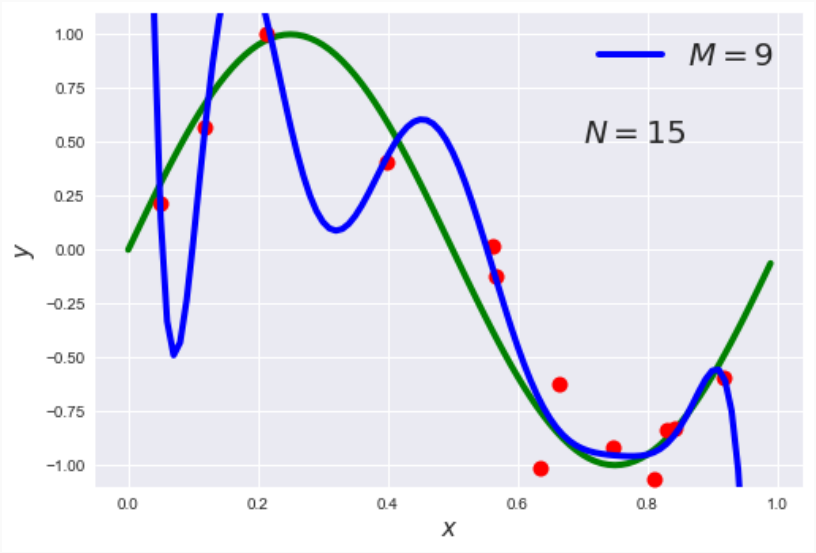


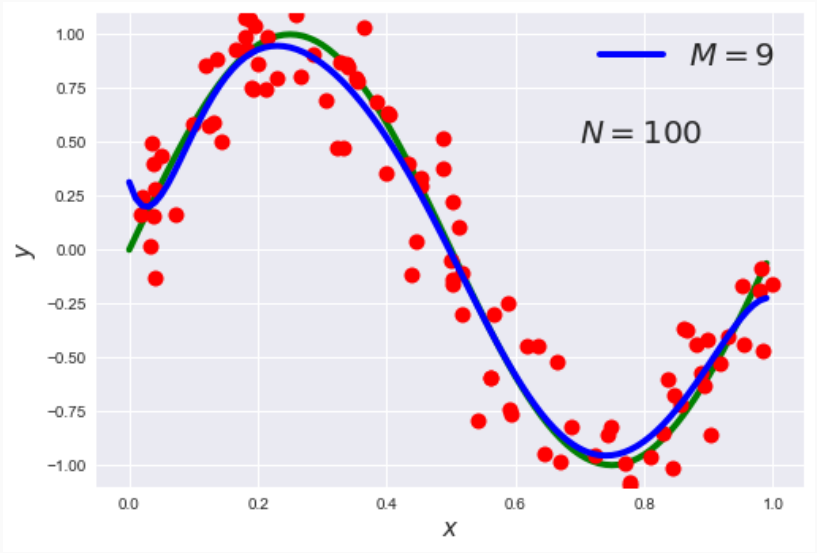


Ideally we want to choose a model that generalizes well, i.e. can provide a good prediction on unseen data.

Underfitting vs overfitting

Note that underfitting/overfitting depends on the data complexity and amount of data.





The $M = 9$ polynomial improves its approximation with more data.

Training loss, testing loss, generalization

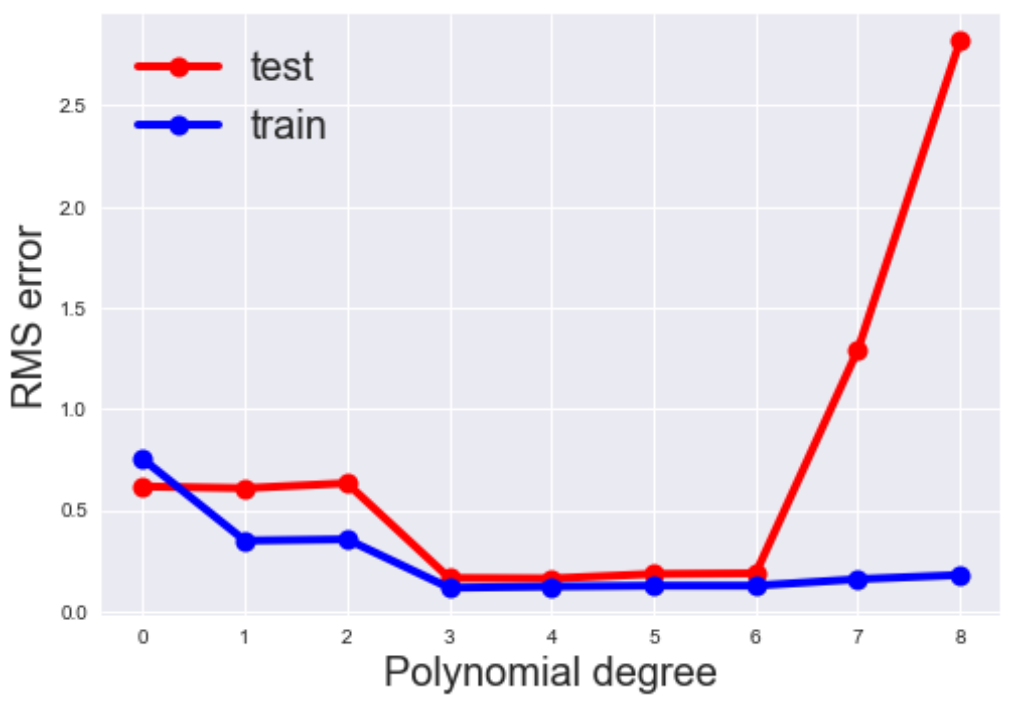
How do we compare these models?

The main idea:

Partition the dataset into 2 disjoint subsets:
a training dataset and a testing dataset.

- Learn the parameters of the model by minimising the average loss on the training dataset.
- Evaluate the generalisation performances of the model by calculating the average loss on the testing dataset.

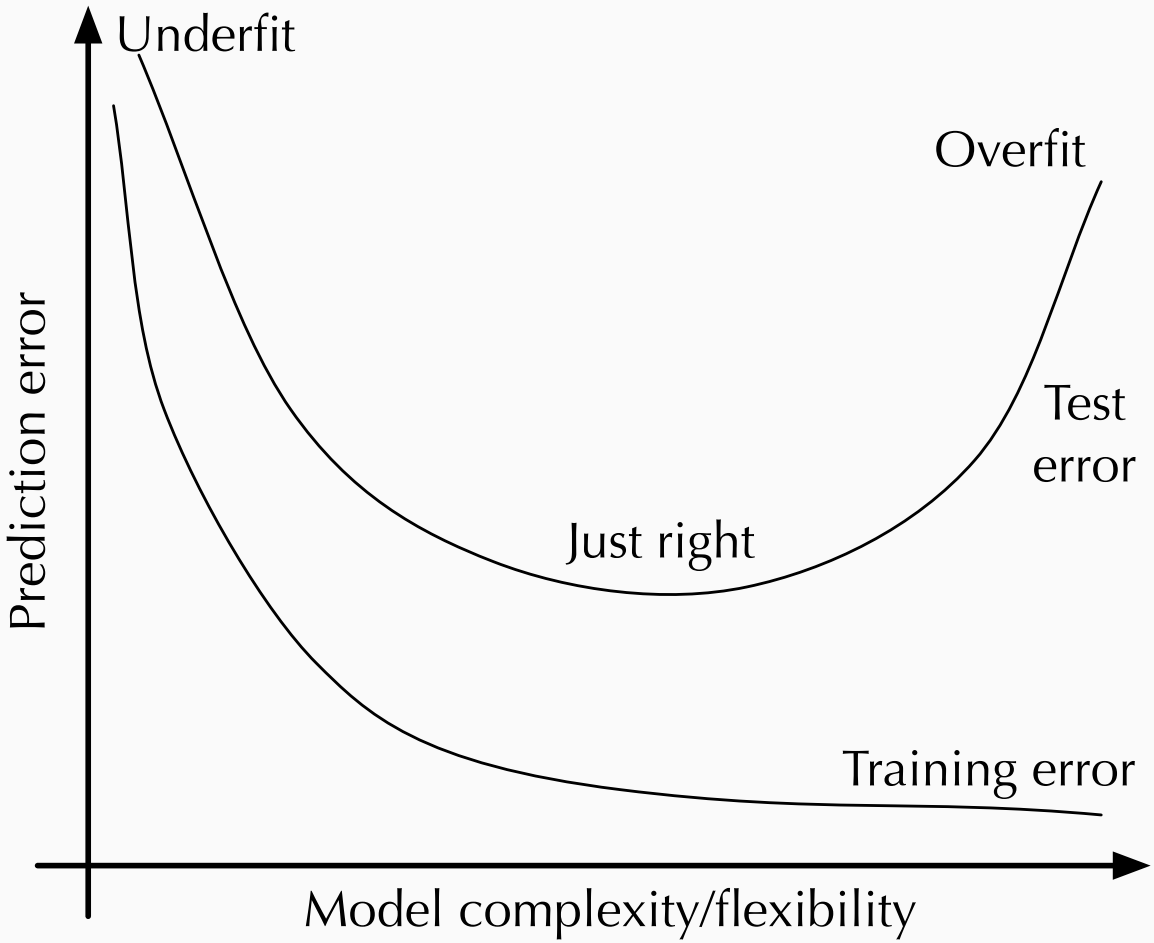
Sinusoidal example



This plot shows a typical behaviour of model performance according to different hyperparameter choices:

- the loss function on the training set decreases with model complexity
- the loss function on the testing set decreases and then increases with model complexity

Underfitting and overfitting: illustration



Training loss, testing loss, generalization

The main idea:

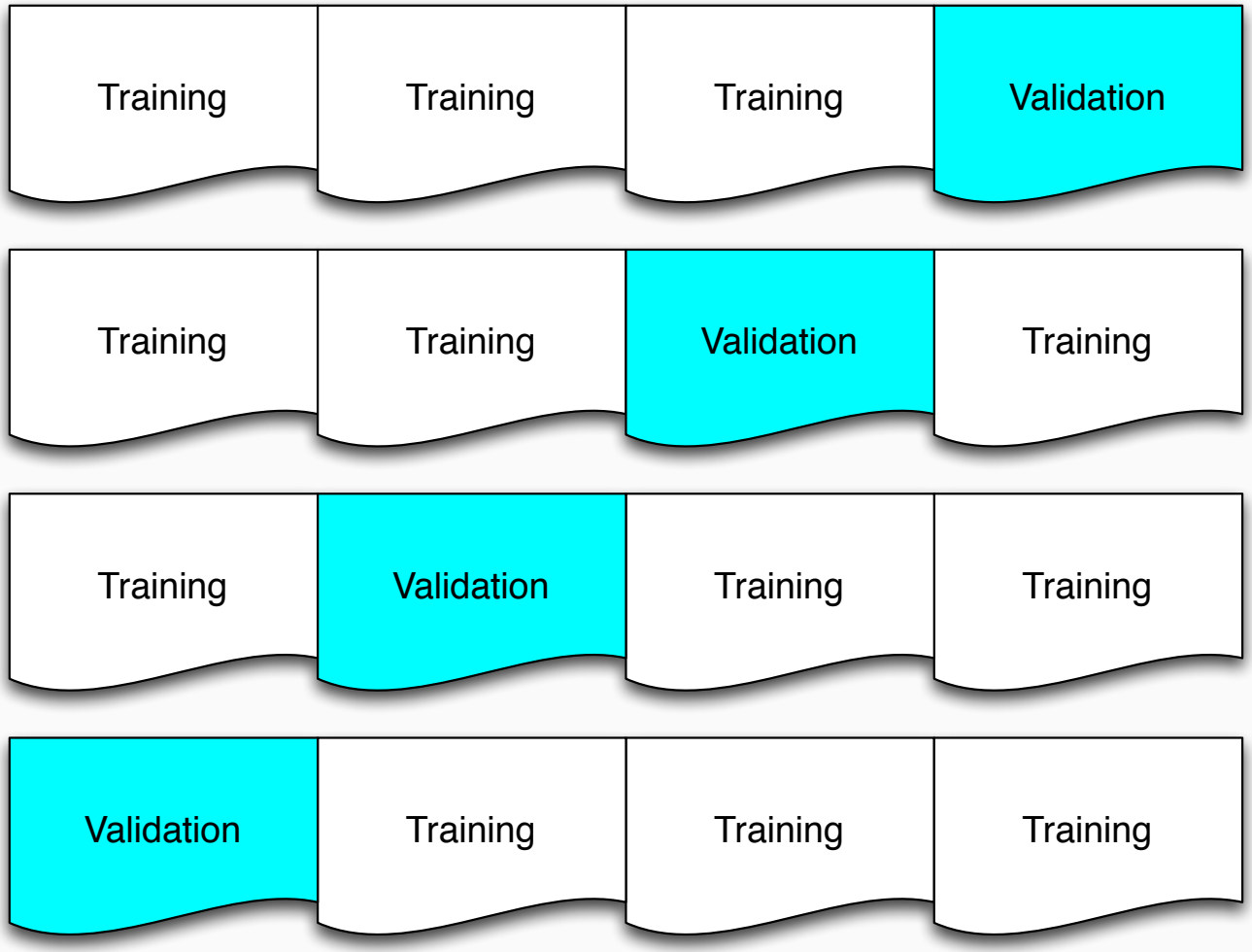
Partition the dataset into 2 disjoint subsets:
a training dataset and a testing dataset.

- Learn the parameters of the model by minimising the average loss on the training dataset.
- Evaluate the generalisation perdormances of the model by calculating the average loss on the testing dataset.

How do we split the dataset into training and testing?
Common approaches: k-fold cross-validation, leave-one-out cross-validation

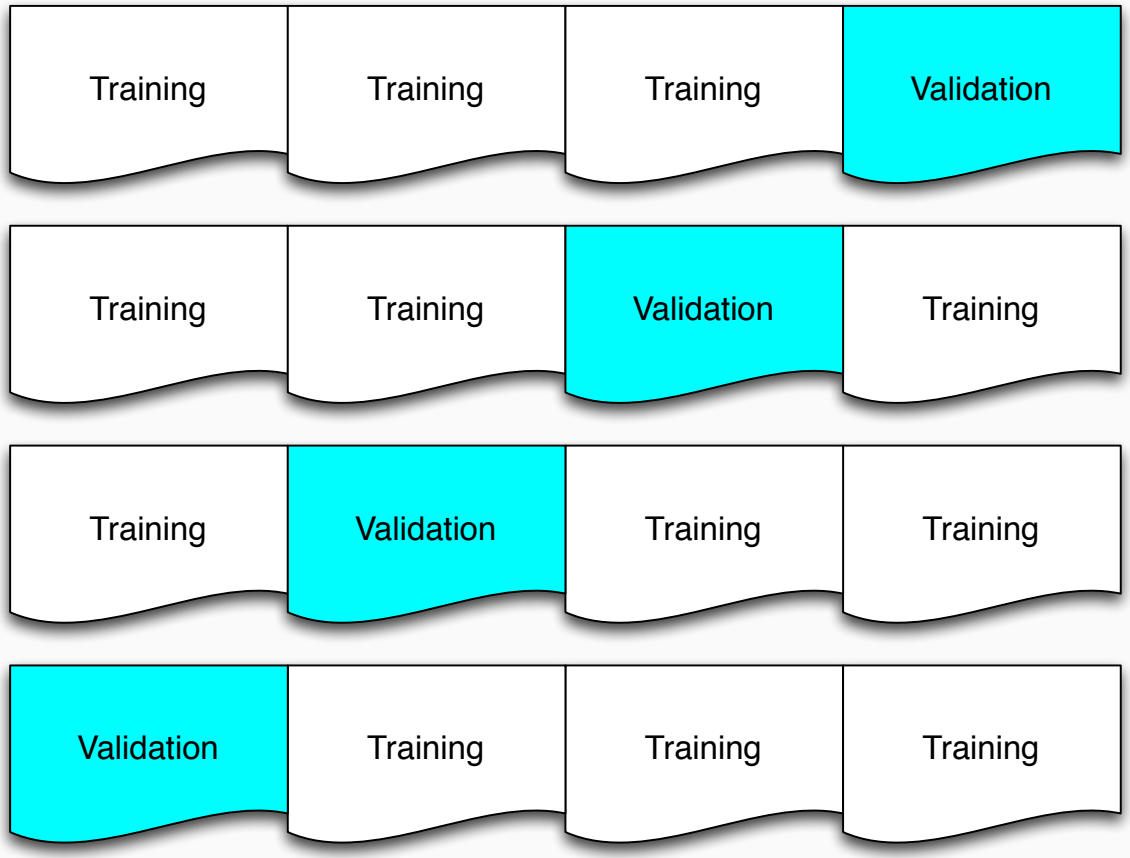
Crossvalidation: basic setup (4 fold crossvalidation)

Randomly split the dataset into 4 folds.



Common metrics to report:
train error and validation error (averaging across folds).

Leave-one-out cross validation



Leave-one-out cross-validation is a special case of K-fold cross-validation where $K = N$, i.e. the test/validation dataset contains only one data point.

Crossvalidation: common mistakes and pitfalls

- Standardizing variables incorrectly between train//test datasets.
- Non-iid data (e.g. time series).

Summary

Today we have developed a feel for the kind of problems that may be tackled using machine learning approaches.

- **Supervised learning** algorithms involve labelled data.
- **Loss functions** can be used to learn the parameters of the models or compare models.
- Underfitting vs overfitting and the use of **cross validation** to compare model performances.

Learning outcomes

At the end of today, you should be able to:

- Know the difference between supervised and unsupervised learning.
- Perform classification using k-nearest neighbours.
- Learn the parameters of a linear model given a dataset.
- Compare different models using a loss function.
- Identify the best model using cross-validation.

Variable selection and regularisation

Body Fat example (from yesterday's tutorial)

Aim: Predict the body fat based on several anthropometric measurements.

Dataset: 71 healthy individuals, 1 output variable and 9 input variables.

During the tutorial yesterday,

- you considered various linear models,
- you inferred their parameters using the given dataset by minimising the mean square error,
- and compared the models using the average leave-one-out cross-validation error.

Could we use another technique to identify which predictors should be included in the model?

Variable selection: which predictors belong in my model?

- Crossvalidation helps us pick models

Variable selection: which predictors belong in my model?

- Crossvalidation helps us pick models
- We can use crossvalidation to decide which predictors to include:

$$f(x) = \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_p x_p$$

Variable selection: which predictors belong in my model?

- Crossvalidation helps us pick models
- We can use crossvalidation to decide which predictors to include:

$$f(x) = \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_p x_p$$

- Considering all possible subsets is NP-hard (exponential in p; but possible for $p \leq 40$)

Variable selection: which predictors belong in my model?

- Crossvalidation helps us pick models
- We can use crossvalidation to decide which predictors to include:

$$f(x) = \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_p x_p$$

- Considering all possible subsets is NP-hard (exponential in p; but possible for $p \leq 40$)
- Classical methods: forward stepwise selection, backward stepwise selection – greedy methods

Variable selection: which predictors belong in my model?

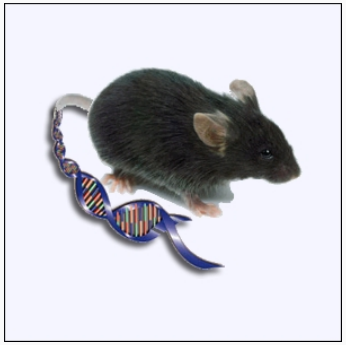
- Crossvalidation helps us pick models
- We can use crossvalidation to decide which predictors to include:

$$f(x) = \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_p x_p$$

- Considering all possible subsets is NP-hard (exponential in p; but possible for $p \leq 40$)
- Classical methods: forward stepwise selection, backward stepwise selection – greedy methods
- Most widely used method for linear models: lasso (least absolute shrinkage and selection operator) and its generalizations: group lasso, elasticnet, and more

Mouse genome dataset

Consider a mouse genome dataset and suppose we want to predict the red blood cell count from SNPs.



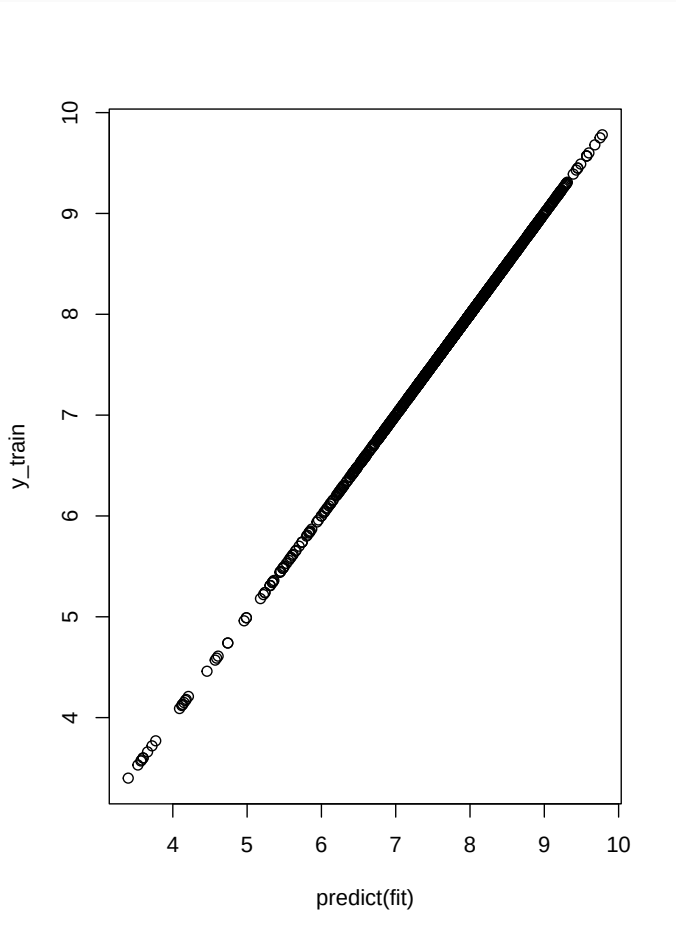
The dataset has

- N=1522 observations
- p= 10346 predictors

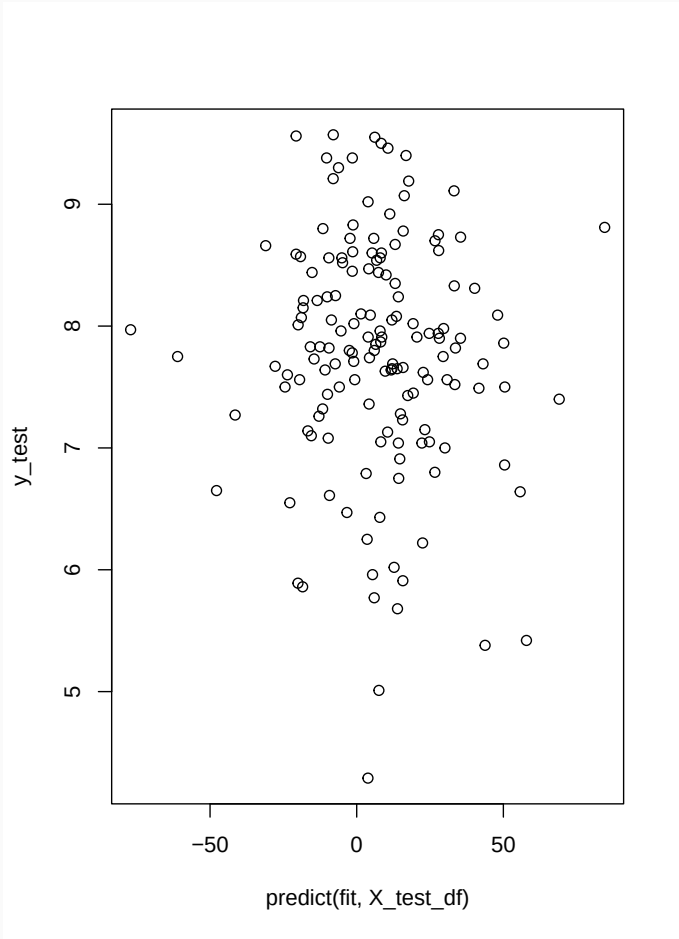
You can read about the dataset at
https://www.sanger.ac.uk/science/data/mouse-genomes-project

Mouse genome dataset

Consider a linear model with 2000 randomly selected predictors (SNPs).
Here are the model predictions:



on training set (N=1217)



on test set (N=152)

The model clearly overfits the dataset here.

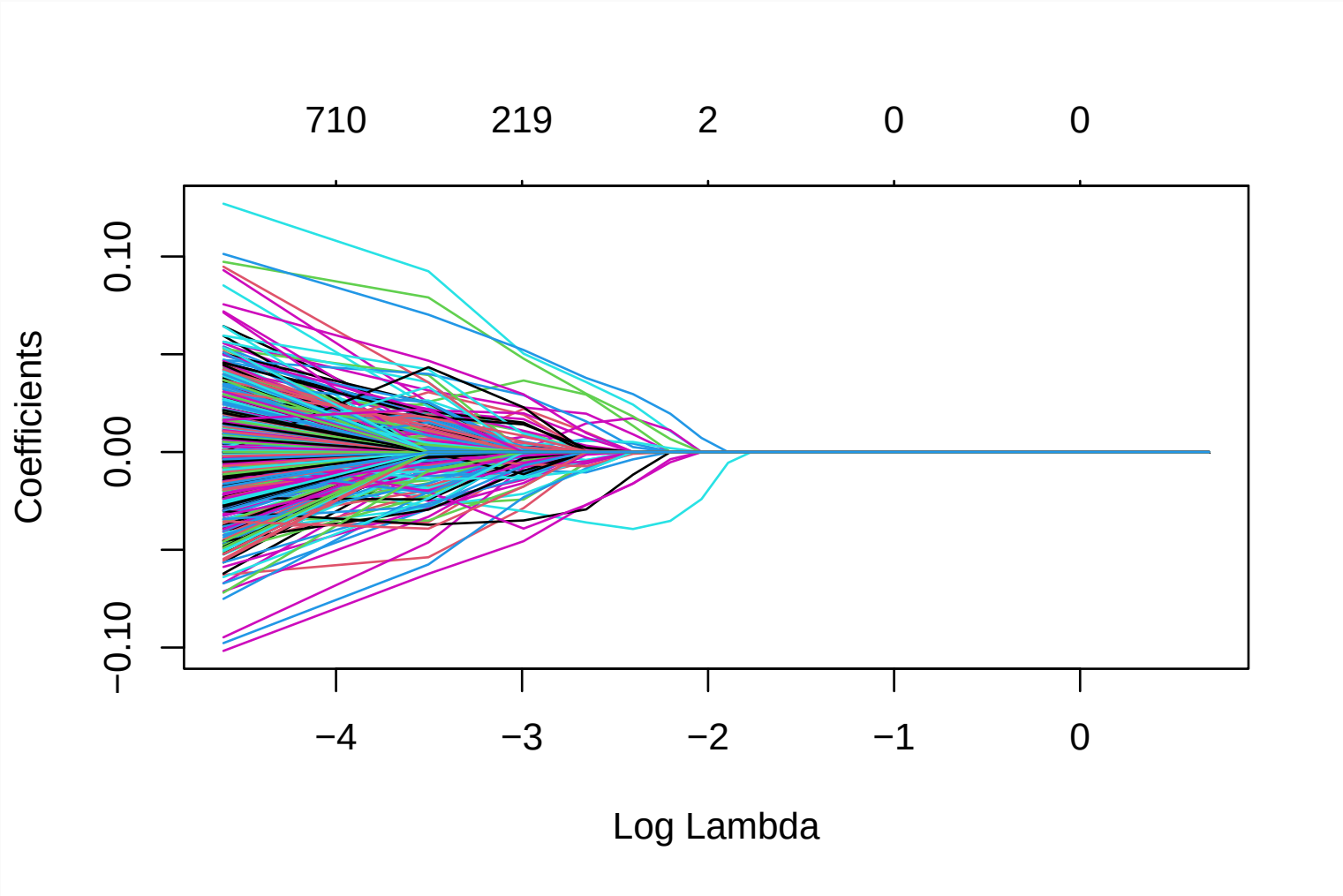
LASSO loss function

The LASSO (least absolute shrinkage and selection operator) approach consists in adding a regularisation term to the loss function:

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^{T}x^{(i)}\right)^{2}+\lambda\sum_{j=1}^{p}|\beta_{j}|\ .$$

Let's minimise this loss function for different values of λ.

Mouse genome dataset – LASSO



LASSO as a variable selection operator

The loss function is

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^{T}x^{(i)}\right)^{2}+\lambda\sum_{j=1}^{p}|\beta_{j}|\ .$$

LASSO as a variable selection operator

The loss function is

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^{T}x^{(i)}\right)^{2}+\lambda\sum_{j=1}^{p}|\beta_{j}|\ .$$

Why is lasso a variable selection operator?

LASSO as a variable selection operator

The loss function is

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^T x^{(i)}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|\ .$$

Why is lasso a variable selection operator?

Constrained optimization problem:

$$\min_{\beta}\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^T x^{(i)}\right)^2$$

$$\text{such that}\sum_{j=1}^{p}|\beta_j|\leq C$$

- For C large (i.e. λ small), $\hat{\beta}=\beta_{\text{OLS}}$.
- For C small (i.e. λ large), many $\hat{\beta}_i=0$.

Hence, variable selection!

Another example

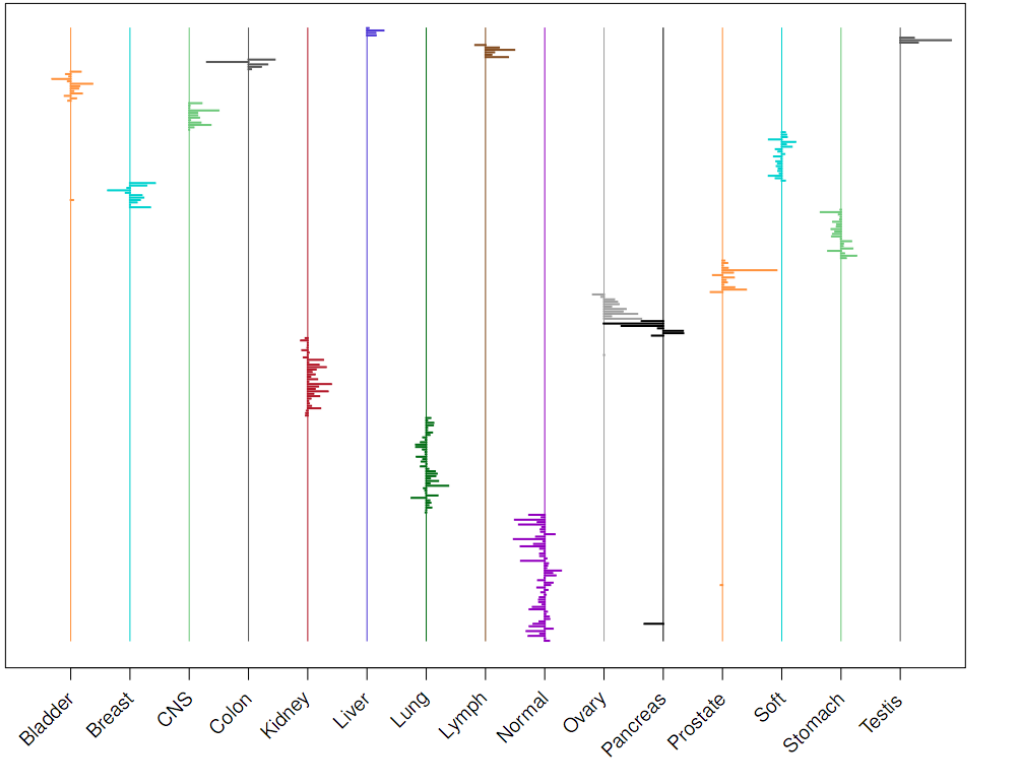


Figure 1.1 *15-class gene expression cancer data: estimated nonzero feature weights from a lasso-regularized multinomial classifier. Shown are the 254 genes (out of 4718) with at least one nonzero weight among the 15 classes. The genes (unlabelled) run from top to bottom. Line segments pointing to the right indicate positive weights, and to the left, negative weights. We see that only a handful of genes are needed to characterize each class.*

Source for figures: *Statistical Learning with Sparsity* by Wainwright, Tibshirani, and Hastie

Regularization: how complex should my model be?

- Flexible models for high-dimensional problems $\rightarrow$ many parameters.
- With many parameters, easy to overfit

Regularization: how complex should my model be?

- Flexible models for high-dimensional problems $\rightarrow$ many parameters.
- With many parameters, easy to overfit
- *Regularization*: Limit flexibility of model to prevent overfitting.
- Add term *penalizing* large values of parameters β:

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \beta^T x^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \beta^T x^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- Also known as *shrinkage* methods—parameters are shrunk towards 0.

Regularization: how complex should my model be?

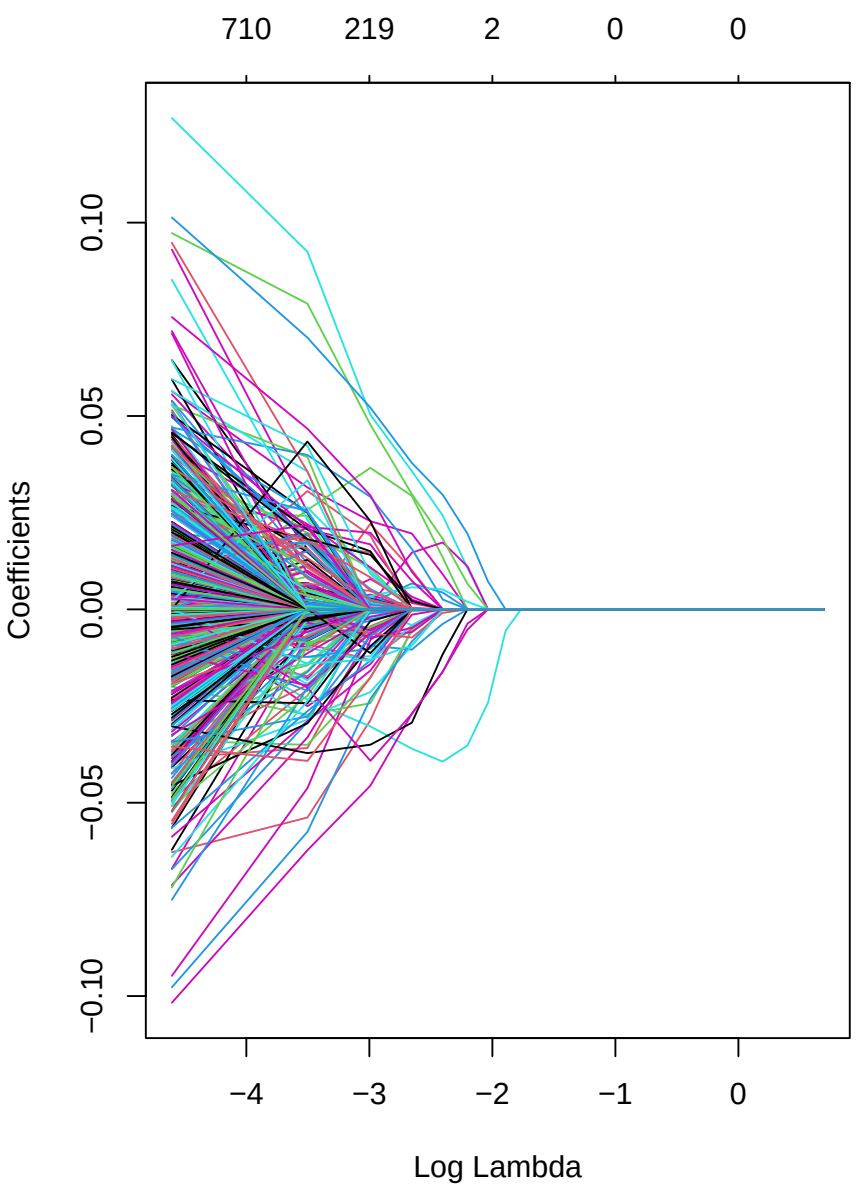
- Flexible models for high-dimensional problems $\rightarrow$ many parameters.
- With many parameters, easy to overfit
- *Regularization*: Limit flexibility of model to prevent overfitting.
- Add term *penalizing* large values of parameters β:

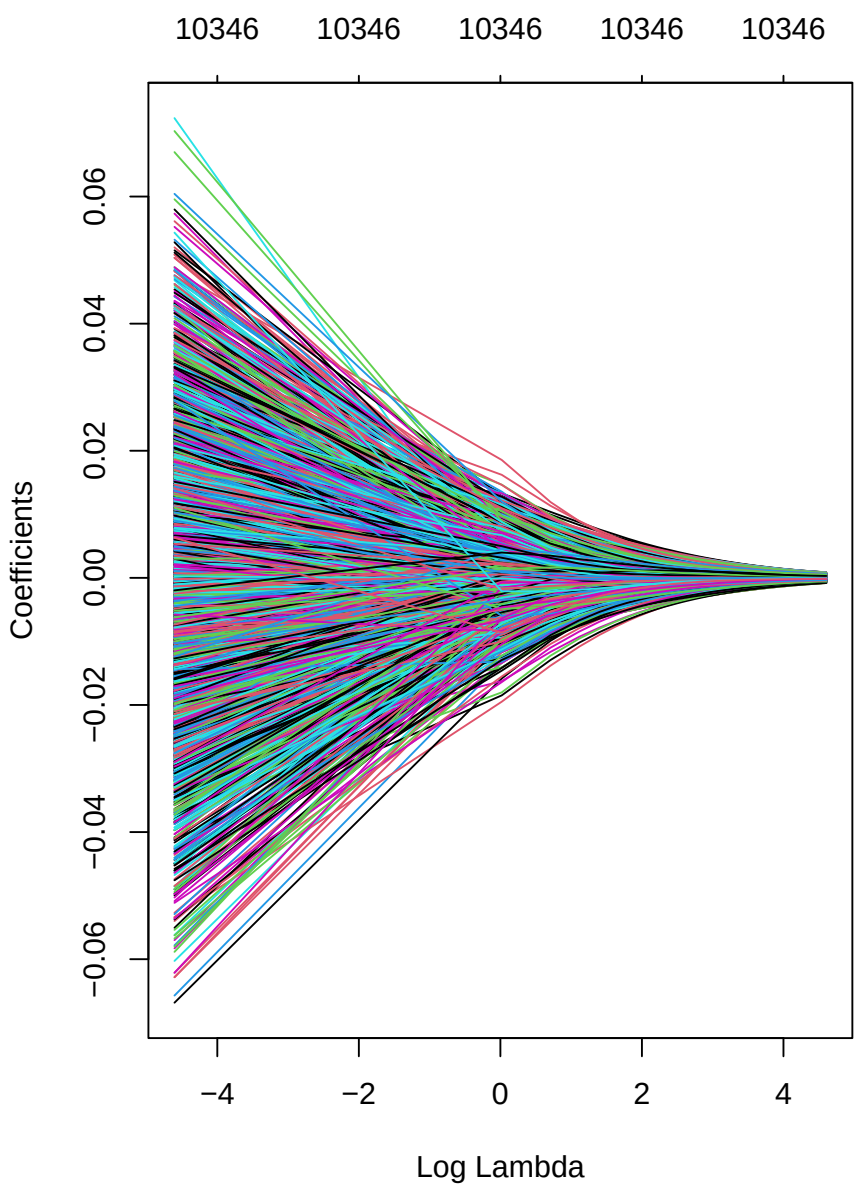
$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \beta^T x^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \beta^T x^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

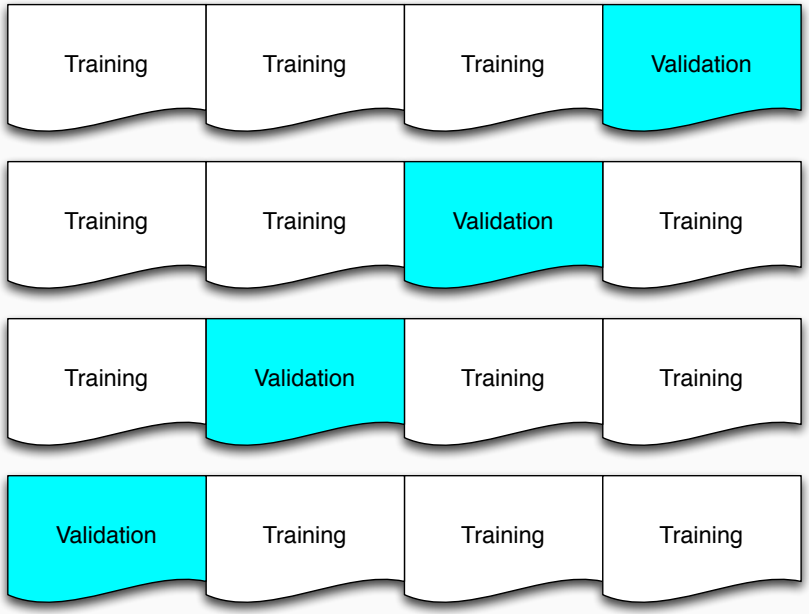
- Also known as *shrinkage* methods—parameters are shrunk towards 0.
- $\lambda \rightarrow \infty$? $\lambda \rightarrow 0$?

Lasso vs. ridge





Hyperparameter choice via cross-validation



1. Randomly split dataset into training and validation sets.
2. For a grid of values of λ
   - Find best $\hat{\beta}$ for each training dataset: minimising loss
   - Calculate the error on the validation dataset
3. Choose λ^* with the smallest average crossvalidated error

The elastic net

Often, very effective to mix both lasso and ridge—the "elastic" net (R package: `glmnet`)

$$\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-\beta^{T}x^{(i)}\right)^{2}+\lambda\left[\alpha\sum_{j=1}^{p}|\beta_j|+(1-\alpha)\frac{1}{2}\sum_{j=1}^{p}\beta_j^2\right]$$

Elastic net combines

- feature elimination from Lasso
- feature coefficient reduction from the Ridge model.

Regularization for high-dimensional settings

Problem: what if $n \ll p$?

- Common in genomics, and many other areas...
- Even if $n > p$ in original dataset, adding d interactions and non-linear feature transformations $\rightarrow n \ll p + d$.

Regularization for high-dimensional settings

Problem: what if $n \ll p$?

- Common in genomics, and many other areas...
- Even if $n > p$ in original dataset, adding d interactions and non-linear feature transformations $\rightarrow n \ll p + d$.

Ridge and lasso solve this problem, too!
Recall that minimising the mean square error, we obtain

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

but when $n \ll p$, $X^T X$ might not be invertible.

Regularization for high-dimensional settings

Problem: what if $n \ll p$?

- Common in genomics, and many other areas...
- Even if $n > p$ in original dataset, adding d interactions and non-linear feature transformations $\rightarrow n \ll p + d$.

Ridge and lasso solve this problem, too!
Recall that minimising the mean square error, we obtain

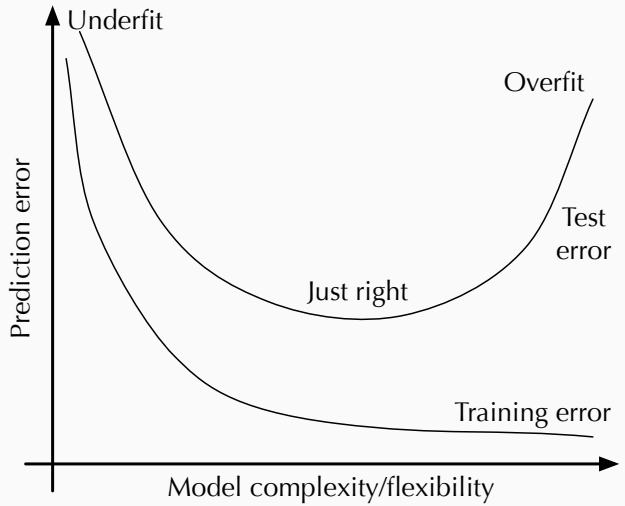
$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

but when $n \ll p$, $X^T X$ might not be invertible.
Minimizing the loss function using a ridge penalty gives

$$\hat{\beta}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y$$

which solves this problems.

Building models: trade between good model and overfitting



Machine Learning model building trades off underfitting/overfitting:

- Building more complex models, e.g. adding more predictors, **adding nonlinear features and additional parameters**, at the expense of overfitting.
- Reducing test error at the expense of increasing training error: simplifying models, using **regularization with lasso/ridge penalties** (or early stoping techniques).