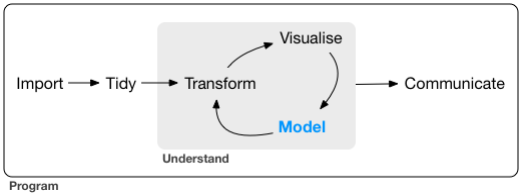
Fitting models

William Lowe

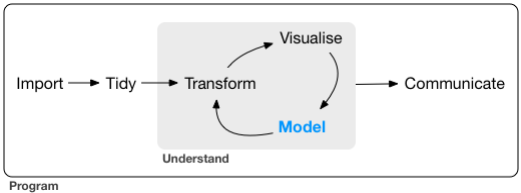
Hertie School

5th November 2020

Plan


Visualise
Import
Tidy
Transform
Communicate
Model
Understand
Program

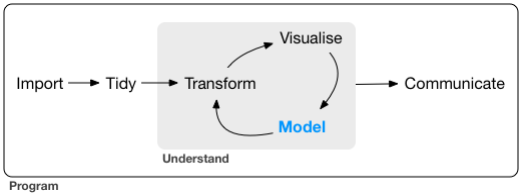
Plan



Model fitting

- → Chapter IV of Wickham and Grolemund (2016)
- → Lots of good advice in that section
- → …which will not age very well

Plan



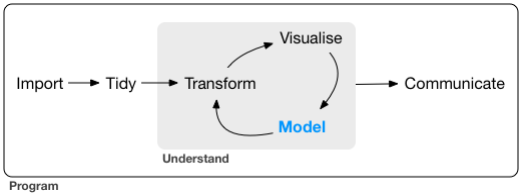
Model fitting

- → Chapter IV of Wickham and Grolemund (2016)
- → Lots of good advice in that section
- → …which will not age very well

This lecture is more about the *big picture*

And the part of model fitting that comes under

Plan



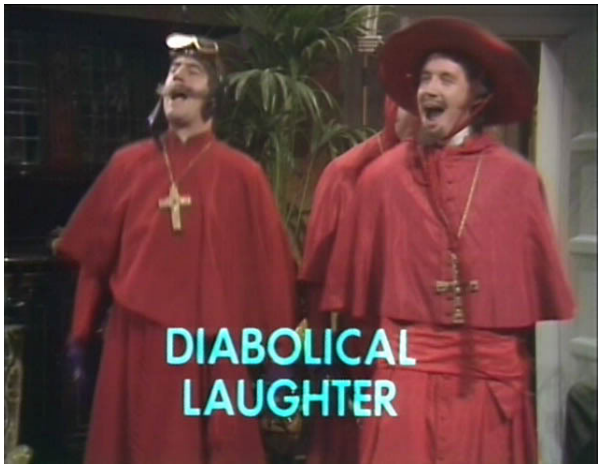
Model fitting

- → Chapter IV of Wickham and Grolemund (2016)
- → Lots of good advice in that section
- → …which will not age very well

This lecture is more about the *big picture*

And the part of model fitting that comes under

Machine Learning


DIABOLICAL
LAUGHTER

Machine Learning in data science

Plan:

- → Types of machine learning and their models
- → Overfitting
- → Regularisation
- → The bias variance decomposition
- → Bayes!
- → Model evaluation and loss functions

Why machine learning

. . . not just models?

Machine Learning (ML) is the part of data analysis that focuses on fitting models

- → Many (all?) your familiar statistical models are special cases
- → But ML has others too
- → It's an almost meaningless term, but it captures the class of things we do fitting models in data science

Machine learning: rather quickly

Inference:

- → Supervised: Learn $P(Y \mid X, Z \ldots)$, or often just its expected value (the 'regression' function)
- → Unsupervised: Learn $P(X, Z)$ (quantitative description)

Machine learning: rather quickly

Inference:

- → Supervised: Learn $P(Y \mid X, Z \ldots)$, or often just its expected value (the 'regression' function)
- → Unsupervised: Learn $P(X, Z)$ (quantitative description)

Action (embeds an inference problem)

- → Reinforcement: Learn a policy $P(\text{Action} \mid \text{State})$ such that the expected future discounted *reward* for the policy's actions is maximized

Machine learning: rather quickly

Inference:

- → Supervised: Learn $P(Y \mid X, Z \ldots)$, or often just its expected value (the 'regression' function)
- → Unsupervised: Learn $P(X, Z)$ (quantitative description)

Action (embeds an inference problem)

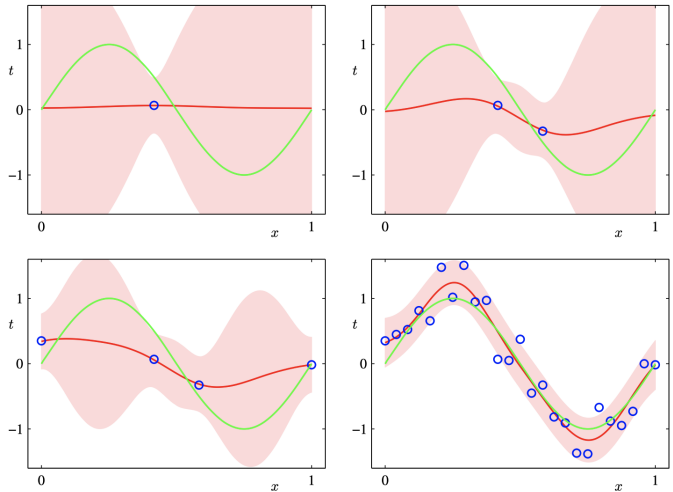
- → Reinforcement: Learn a policy $P(\text{Action} \mid \text{State})$ such that the expected future discounted *reward* for the policy's actions is maximized

We'll be interested in supervised, traditionally separated into

- → Regression: usually implicitly assumes symmetric constant ϵ (or doesn't have an opinion…)
- → Classification: ambiguous between *choosing* one of K classes and estimating $P(Y = k \mid X, Z \ldots)$

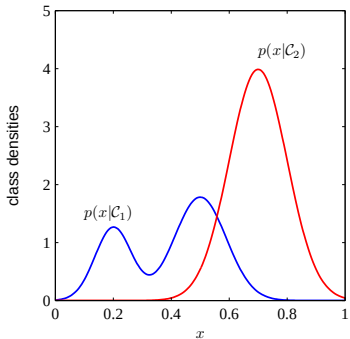
In any case, both go for $\mathrm{E}[Y \mid X, Z \ldots]$

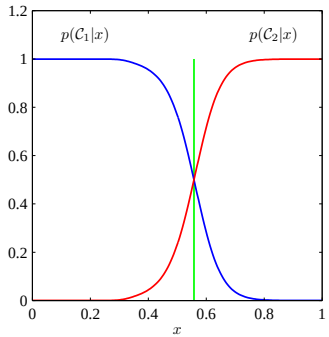
Regression flexibility



Classifier flexibility

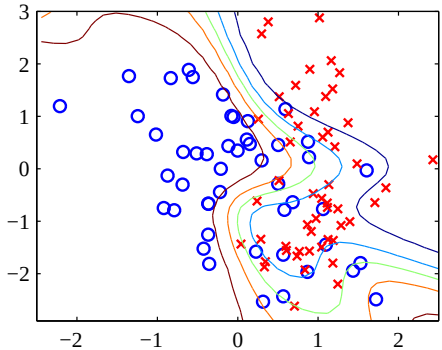
Simple models: simple decision boundaries





Classifier flexibility

More flexible models: more complicated decision boundaries



Flexibility

You could, if you like, think of linear regression and logistic regression in each of these categories

→ It's illuminating to do so (see the first few chapters of Bishop, 2006)

So what's the ML model difference?

→ More flexible forms for $\mathrm{E}[Y \mid X, Z \ldots]$
→ Higher dimensional predictors, i.e. lots more $X, Z \ldots$

Many ML regression models will embed a more familiar model, e.g. logistic regression inside neural networks

Others will start from scratch and build $\mathrm{E}[Y \mid X, Z \ldots]$ differently, e.g. classification trees

Indifference

As an engineering tool, ML models will seldom care about what X, Z etc. actually are, or distinguishing one parameter among the others

Indeed most are *non-parametric*

→ Reminder: 'non-parametric' does not mean 'does not have parameters, it means 'has so many parameters that I do not care to know them by name'

Key insight:

→ It's better to fit an infinitely flexible model and figure out how to constrain it than to fit a too simple model and figure out how to make it fit better
→ We may as well *start* with universal function approximators (Hornik et al., 1989)

Key problem:

→ How to constrain it?

Breaking regression

What happens when there are more variables than cases?

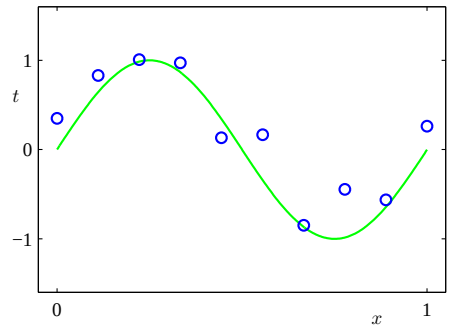
→ Regular regression breaks

What happens when you add all the squares and cubes and interactions as predictors?

→ Standard errors explode; same amount of data, but more parameters to learn from it.
→ Generalization to new data gets worse; now we can fit everything better, we fit noise better

These are the same problem in different degrees

The problem, with polynomials



For consistency with Bishop ch. 1 let's call

- → the outcome t_n
- → the regression coefficients $w_j \in \mathbf{w}$ ('weights')
- → our estimate of the expected value of t_n, $\hat{t}_n = y(x, \mathbf{w})$

Adding flexibility

Consider polynomial models of t. We'll fit / make predictions like this:

$$
\begin{aligned}
y(x, \mathbf{w}) &= w_0 \\
&= w_0 + w_1 x \\
&= w_0 + w_1 x + w_2 x^2 \\
&= w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M \\
&= \sum_{j}^{M} w_j x^j
\end{aligned}
$$

Adding flexibility

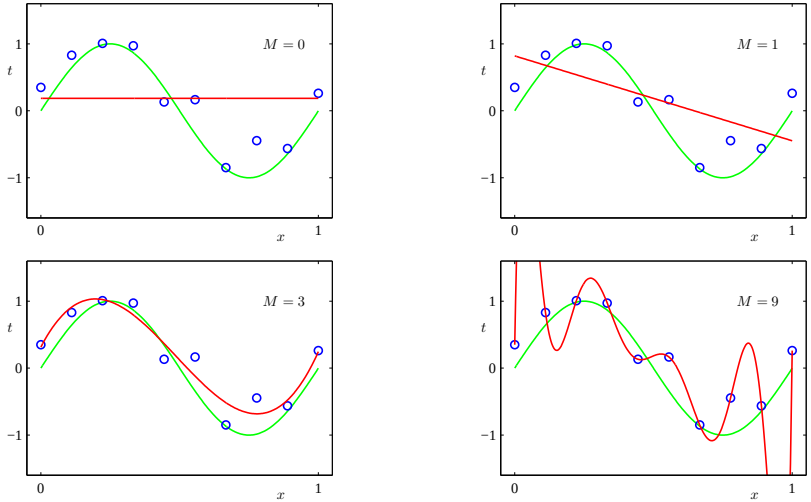
Consider polynomial models of t. We'll fit / make predictions like this:

$$
\begin{aligned}
y(x, \mathbf{w}) &= w_0 \\
&= w_0 + w_1 x \\
&= w_0 + w_1 x + w_2 x^2 \\
&= w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M \\
&= \sum_{j}^{M} w_j x^j
\end{aligned}
$$

The *flexibility* of this model is driven by M, which we can think of as determining the model class

→ Roughly: the set of functions that can be represented

Adding flexibility

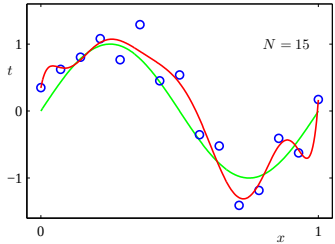


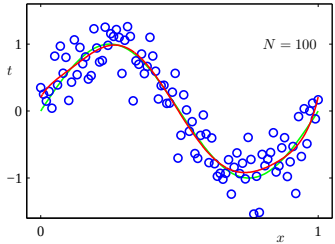
Adding flexibility

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	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

Overfitting

Things are not so bad when there is more data (here M=9)

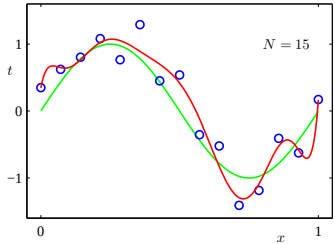


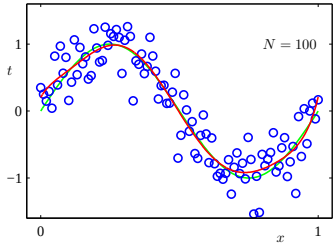


But there isn't always going to be more data…

Overfitting

Things are not so bad when there is more data (here M=9)



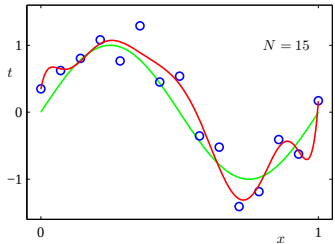


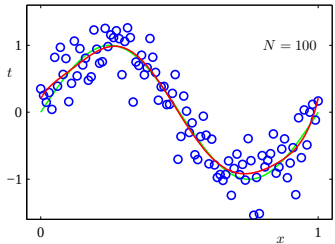
But there isn't always going to be more data…

However, we can keep all M, i.e. maintain the flexibility in the model class, *if* we can constrain the size of the weights

→ This calls for a *hyperparameter*, a parameter that controls other parameters

Overfitting





When there's lots of persuasive data:

→ override the hyperparameter and make use of the model flexibility

When there isn't,

→ keep weights small, and therefore the function smooth

Regularization by hyperparameter

Here, we're fitting the model (maximizing the likelihood) using OLS, which *minimises* the sum of squared errors

$$E_{\text{OLS}} = \frac{1}{2}\sum_{n}^{N}(y(x_n, \mathbf{w}) - t_n)^2$$

Note: minimising error rather than maximizing the likelihood is the way ML people think about things (hence, no minus sign)[1]

[1] The 1/2 is there to hint that this is the log likelihood for a Normal distribution (with constant error variance, so it doesn't matter to E)

Regularization by hyperparameter

Here, we're fitting the model (maximizing the likelihood) using OLS, which *minimises* the sum of squared errors

$$E_{\text{OLS}} = \frac{1}{2}\sum_{n}^{N}(y(x_n, \mathbf{w}) - t_n)^2$$

Note: minimising error rather than maximizing the likelihood is the way ML people think about things (hence, no minus sign)[1]

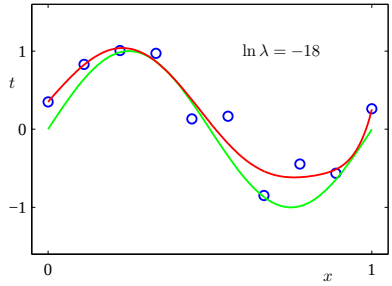
Let's keep that plan, but add an extra term to control the weights

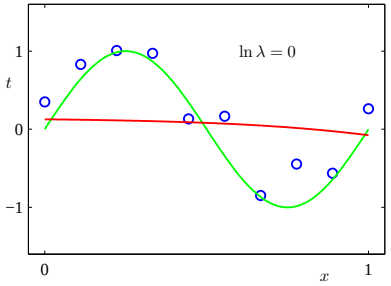
$$E_\lambda = \frac{1}{2}\sum_{n}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_{m}^{M} w_m^2$$

and a hyperparameter λ to say how seriously we should take it as an error component

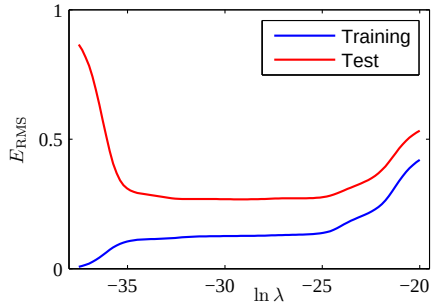
[1] The 1/2 is there to hint that this is the log likelihood for a Normal distribution (with constant error variance, so it doesn't matter to E)

Consequences





The sweet spot



→ The left extreme is $\lambda = 0$ (no regularization)

→ The right extreme is all zero weights (predict of 0 for every point)

→ With fixed data, decreasing λ allows more of the model class's inherent flexibility to show

Choosing hyperparameter values

We can't fit λ by minimising the sum of squares

→ That would just set it to zero (why?)

Choosing hyperparameter values

We can't fit λ by minimising the sum of squares

→ That would just set it to zero (why?)

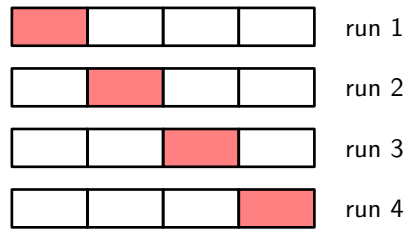
One reliable option is crossvalidation (CV)

→ Make a grid of hyperparameter values

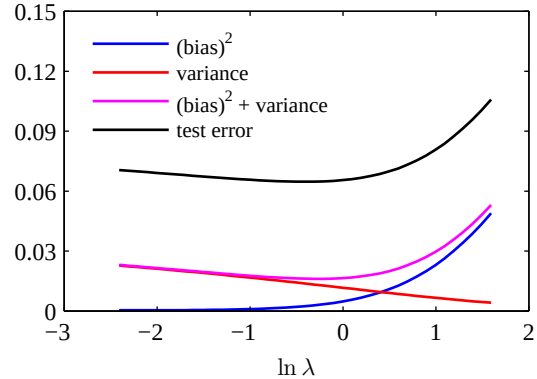
→ Randomly divide the data set into 4 (or some other value > 1)

→ For each hyperparameter value, train a model on white and test on red

→ Choose the hyperparameter value that minimizes the average error on reds



Bias and Variance



Error is unavoidable but bias and variance trade off

Thinking about Inference

Let's revisit this tricky looking error function

$$E_\lambda = \frac{1}{2}\sum_n^N (y(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_m^M w_m^2$$

We motivated this by saying that minimizing it kept the weights small, which kept the function smooth.

There's another way to motivate it: as a *Bayesian inference*

Thinking about Inference

Let's revisit this tricky looking error function

$$E_\lambda = \frac{1}{2}\sum_n^N (y(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_m^M w_m^2$$

We motivated this by saying that minimizing it kept the weights small, which kept the function smooth.

There's another way to motivate it: as a *Bayesian inference*

Bayesian inference is a theory of *learning under uncertainty*, mostly normative but in many interesting cases also descriptive, and a very popular way to 'fit' a model.

Bayes for model fitting

Conceptually simple:

→ Just use probability theory for everything

Practically often quite hard!

Bayes for model fitting

Conceptually simple:

→ Just use probability theory for everything

Practically often quite hard!

Recall Bayes theorem

$$P(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) = \frac{P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) P(\mathbf{w} \mid \mathbf{x})}{P(\mathbf{t} \mid \mathbf{x})}$$

but $\mathbf{w}$ doesn't depend on X, so we'll drop that conditioning

$$P(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) = \frac{P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) P(\mathbf{w})}{P(\mathbf{t} \mid \mathbf{x})}$$

Bayesian inference for model parameters

$$P(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) = \frac{P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) P(\mathbf{w})}{P(\mathbf{t} \mid \mathbf{x})}$$

The *prior:* the distribution of plausible values for the weights

→ $P(\mathbf{w})$

The *likelihood*: how the data is made, assuming we have values for the weights

→ $P(\mathbf{t} \mid \mathbf{x}, \mathbf{w})$

The *posterior*: the distribution of plausible values for the weights, in the light of the data

→ $P(\mathbf{w} \mid \mathbf{x}, \mathbf{t})$

Some candidate distributions

Assume everything is Normal distributed with constant variance…

Some candidate distributions

Assume everything is Normal distributed with constant variance…

$$P(\mathbf{w}) \propto \exp\left(-\sum_{n}^{9} \frac{\alpha}{2} w_n^2\right)$$

Normal with zero mean, and variance α^{-1}

Assuming independent data points, we can do the same for the ts

$$P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) \propto \exp\left(-\sum_{n}^{N} \frac{\beta}{2} (y(x_n, \mathbf{w}) - t_n)^2\right)$$

which is Normal with mean $y(x_n, \mathbf{w})$ and variance β^{-1}

The *posterior* distribution for $\mathbf{w}$ is just these two multiplied together and divided by a constant

→ So one way to estimate the best set of $\mathbf{w}$ is the choose the ones that have maximum posterior probability (MAP)

Inference for the parameters

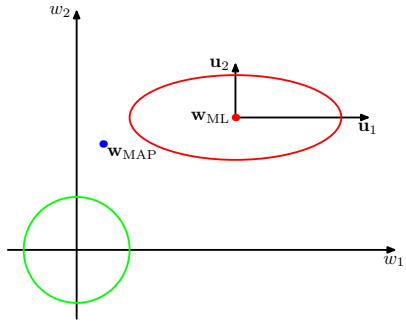
$$P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) \propto \exp\left(-\sum_{n}^{N} \frac{\beta}{2}(y(x_n, \mathbf{w}) - t_n)^2\right) \qquad P(\mathbf{w}) \propto \exp\left(-\sum_{n}^{9} \frac{\alpha}{2} w_n^2\right)$$

Another way to do the same thing is to maximize the *log* of the posterior distribution

$$\begin{aligned}
\log P(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) &\propto \log\left(P(\mathbf{t} \mid \mathbf{x}, \mathbf{w})\ P(\mathbf{w})\right) \\
&\propto \log P(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) + \log P(\mathbf{w}) \\
&\propto -\left[\sum_{n}^{N} \frac{1}{2}(y(x_n, \mathbf{w}) - t_n)^2 \ + \ \sum_{n}^{9} \frac{\lambda}{2} w_n^2\right] \\
&= -E_\lambda
\end{aligned}$$

which is the same as *minimising* E_λ when $\lambda = \alpha/\beta$

Model bias



- → $\mathbf{w}_{\mathrm{ML}}$ minimizes $E_{\lambda=0} = E_{\mathrm{OLS}}$
- → The origin minimizes $E_{\lambda=\infty}$
- → $\mathbf{w}_{\mathrm{MAP}}$ minimizes E_λ when we set λ sensibly to balance the two parts of the error function
- → Alternatively, the information source we are more confident about (Bayes)

Inference for everything else

So we've got a whole posterior distribution over $\mathbf{w}$

→ Or just the very peak of it if we want to assign a definite value to each weight

But that also means we've got an *implied* posterior distribution for everything that depends on $\mathbf{w}$

→ the probability that $w_3 > w_2$ or any other arbitrary relationship
→ predictions from new (or old) $\mathbf{x}$

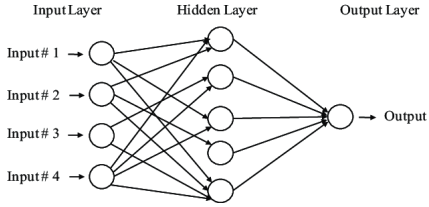
Do we need to do more math to figure out what it is? Not if we can get a *sample* from the posterior

$$t_{n+1} \sim \int P(t_{n+1} \mid x_{n+1}, \mathbf{w}) P(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) \, d\mathbf{w}$$

but with a S samples $\mathbf{w}^{(i)}$ from $P(\mathbf{w} \mid \mathbf{x}, \mathbf{t})$

$$\mathrm{E}[t_{n+1}] = \frac{1}{S} \sum_{i}^{S} P(t_{n+1} \mid x_{n+1}, \mathbf{w}^{(s)})$$

Neural networks



A multilayer perceptron (MLP) with 1 hidden layer f J 'units' for D-dimensional input data x is

$$y(x, \mathbf{w}) = \sum_{j}^{J} w_j \phi_j(x, \mathbf{w}^{(j)})$$

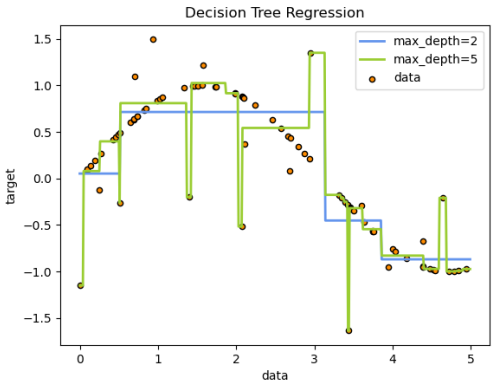
where ϕ_j is some nonlinear function of the input data, e.g.

$$\phi_j(x, \mathbf{w}^{(j)}) = 1/(1 + \exp(-\sum_{d} w_d^{(j)} x_d)$$

That's a universal approximator (Hornik et al., 1989) that needs serious regularization

And now for something totally different

Alternatively, we can change the definition of $P(t \mid x, \mathbf{w})$ altogether, e.g. regression trees (from scikit-learn documentation)



Blue tree (2 levels)

```
if x > 3.2 then
  if x > 3.9 then -0.9 else -0.5
else
  if x > 0.5 then 0.8 else 0.1
```

The green tree allows up to 5 levels, and overfits

Regression trees

For regression trees, **w** are now the split positions and (one) hyperparameter is the depth of the tree

→ so constraining that adds bias and reduces variance

In general we can also prevent overfitting by bagging (Breiman, 1996) or variations on that theme (e.g. Random forests Cutler et al., 2012)

→ bootstrapping the dataset
→ Fitting trees to each bootstrap sample
→ Averaging the resulting predictions

Regression trees

For regression trees, **w** are now the split positions and (one) hyperparameter is the depth of the tree

→ so constraining that adds bias and reduces variance

In general we can also prevent overfitting by bagging (Breiman, 1996) or variations on that theme (e.g. Random forests Cutler et al., 2012)

→ bootstrapping the dataset
→ Fitting trees to each bootstrap sample
→ Averaging the resulting predictions

Or we can be Bayesians again (e.g. Chipman et al., 2010, BART)

→ Start with a prior forest
→ See the data
→ Prune down to a posterior forest

Bias as a good thing

Clearly regularization generates bias. Seems like a bad thing…

But it's necessary

- → All models are wrong; some are useful
- → The *No Free Lunch theorem* (Wolpert, 1996) says that averaged over all possible problems, no learning algorithm is better than any other
- → Happily we don't deal with all possible problems, so we can and should choose a model bias to fit the problem

And helpful

- → It's how we get less variance

And annoying because it slows convergence

This is better than the alternative, which is not being consistent and not knowing it

Model evaluation

Strategy:

- → Out-of-sample testing is the gold standard
- → In-sample fit is…ok
- → Cross-validation is better

Tactics: operationalising performance

- → For regression models: Mean squared error is a common standard
- → Expected utility is also possible but requires a *loss function*
- → often obviously important for classification tasks

Classification

We've been assuming a regression context for out ML so far, but we can also think about classification

Reminder: classification is two things, often confused. In a simple two class (0/1) classification

- → Estimating $E(Y \mid X_1 \dots X_K) = P(Y = 1 \mid X_1 \dots X_K)$
- → Deciding 1 or 0 in the light of $P(Y = 1 \mid X_1 \dots X_K)$

Implicitly you may be used to deciding 1 if $P(Y = 1 \mid X_1 \dots X_K) > 0.5$

However, it is often more costly to mistake a 1 for a 0 than a 0 for a 1, e.g.

- → 1 means a state will collapse in the next year (e.g. King & Zeng, 2001)
- → The losses are far from equal
- → Intuitively we should require lower probability to choose 1 when mistaking 1 for 0 is very costly

Classification

Decision theory:

- → L_{ij} is the cost of mistaking i for j e.g. L_{10} is the cost of mistaking a 1 for a 0
- → Minimize the expected L by choosing the i that minimizes

$$\sum_j L_{ij} P(Y = i \mid X_1 \dots X_K)$$

For 1/0 decisions another way to put this is in terms of a cutoff: Choose

$$\hat{Y} = \begin{cases} 1 & \text{if } P(Y = 1 \mid X_1 \dots X_K) > \frac{1}{1+C} \\ 0 & \text{otherwise} \end{cases}$$

where

$$C = \frac{L_{10}}{L_{01}}$$

Classification errors

From the loss function we can also identify two sorts of error

- → Mistaking a 1 for a 0: $P(\hat{Y} = 0 \mid Y = 1)$
- → Mistaking a 0 for a 1: $P(\hat{Y} = 1 \mid Y = 0)$

A useful and closely related pair of quantities are

$$P(\hat{Y} = 1 \mid Y = 1) = 1 - P(\hat{Y} = 0 \mid Y = 1) \qquad \text{(recall)}$$

$$P(Y = 1 \mid \hat{Y} = 1) = \frac{P(\hat{Y} = 1 \mid Y = 1)P(Y = 1)}{P(\hat{Y} = 1)} \qquad \text{(precision)}$$

Varying C expresses a tradeoff between these too

- → High C lowers the cutoff, which increases recall but decreases precision
- → Low C raises the cutoff which increases precision but decreases recall

Unknown losses, unknown tradeoffs

Sometimes we don't have (or can't commit to) some loss matrix L or a prefered balance between precision and recall

However, since each value of C implies such a loss / balance, we can ask how well a classifier does for *all possible* cutoffs

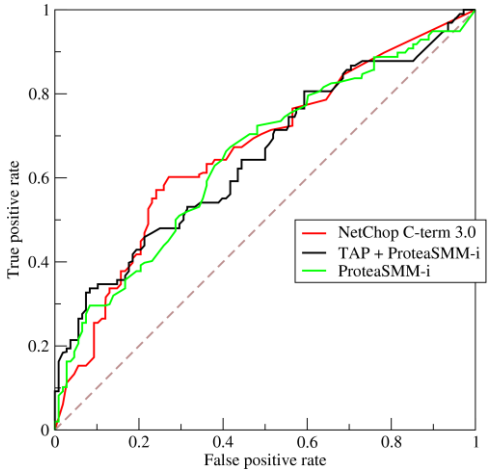
Traditionally we plot precision and recall in a *Receiver Operating Characteristic* (ROC) curve for a wide range of cutoffs

Warning:

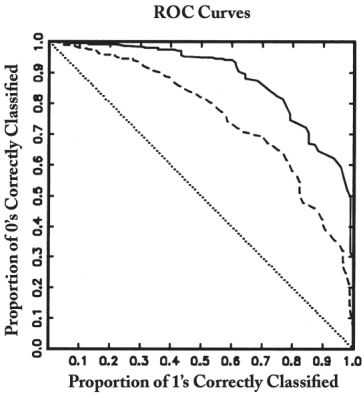
→ All these things are related, so some authors prefer different pairs of performance quantities [sigh]

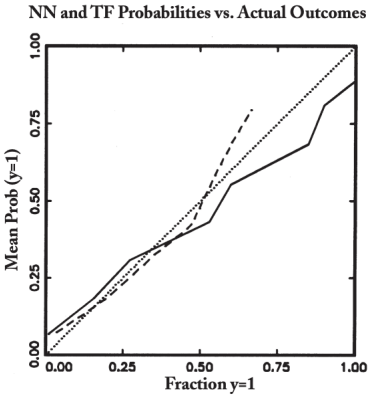
Traditionally, ROC curves plot recall and 1-precision

Receiver Operating Characteristic curves



ROC AND CALIBRATION





Model fitting in data science

Plan:

- → Types of machine learning
- → Overfitting is the problem
- → Regularization as the cure
- → Bias-variance decomposition applies everywhere
- → Bayes, now you know you want to
- → Model evaluation is not optional!

References

Bishop, C. M. (2006). 'Pattern recognition and machine learning'. Springer.

Breiman, L. (1996). 'Bagging predictors'. *Machine Learning*, *24*(2), 123–140.

Chipman, H. A., George, E. I. & McCulloch, R. E. (2010). 'Bart: Bayesian additive regression trees'. *The Annals of Applied Statistics*, *4*(1), 266–298.

Cutler, A., Cutler, D. R. & Stevens, J. R. (2012). Random forests. In C. Zhang & Y. Ma (Eds.), *Ensemble machine learning* (pp. 157–175). Springer US.

Hornik, K., Stinchcombe, M. & White, H. (1989). 'Multilayer feedforward networks are universal approximators'. *Neural Networks*, *2*, 359–366.

King, G. & Zeng, L. (2001). 'Improving forecasts of state failure'. *World Politics*, *53*(4), 623–658.

Wickham, H. & Grolemund, G. (2016). 'R for data science: Import, tidy, transform, visualize, and model data' (First edition). O'Reilly
OCLC: ocn968213225.

References

Wolpert, D. H. (1996). 'The lack of a priori distinctions between learning algorithms'. *Neural Computation*, *8*, 1341–1390.