# Machine Learning

William Lowe

Hertie School

27th September 2020

## Machine learning: rather quickly

Plan

- → Types of machine learning
- → Going further than regression
- → More flexibility with polynomials
- → Overfitting and regularization
- → Bias variance decomposition
- → The good part about bias
- → Example ML models: Ridge, lasso, trees, neural networks
- → Back to causal inference in the style of Frisch, Waugh, and Lovell
- → Double/debiased ML, the very idea

## Machine learning: rather quickly

Inference:

- → Supervised: Learn $P(Y \mid X, Z \ldots)$, or often just its expected value
- → Unsupervised: Learn $P(X, Z)$

## Machine learning: rather quickly

Inference:

- → Supervised: Learn $P(Y \mid X, Z \ldots)$, or often just its expected value
- → Unsupervised: Learn $P(X, Z)$

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 $P(Y \mid X, Z \dots)$, or often just its expected value
- → Unsupervised: Learn $P(X, Z)$

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 $\epsilon$ (or doesn't have an opinion...)
- → Classification: ambiguous between *choosing* one of $K$ classes and estimating $P(Y = k \mid X, Z \dots)$

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

## 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 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. 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’

Unsurprisingly, this part of ML came late to causal inference

## Extending regression

What happens when there are more variables than cases?

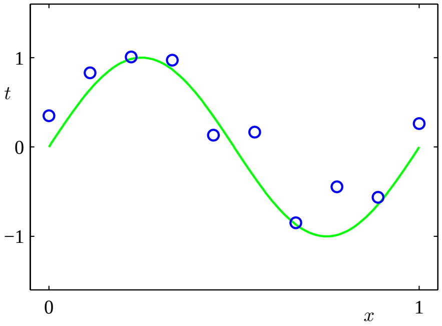
- → Regular regression breaks

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

## Adding 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

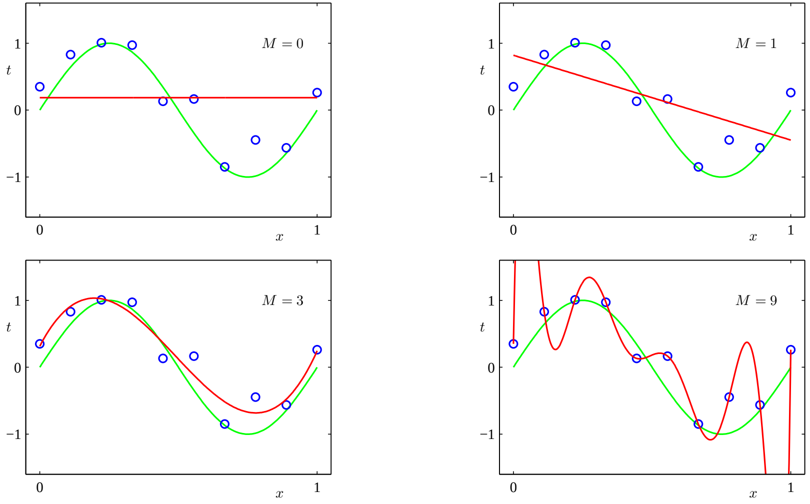
Consider polynomial models of $t$. We'll fit / make predictions like this:

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

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



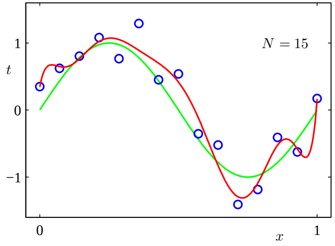
[illegible]

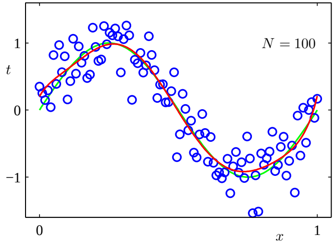
## Adding flexibility

| | $M=0$ | $M=1$ | $M=6$ | $M=9$ |
|---|---|---|---|---|
| $w_0^\star$ | 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)

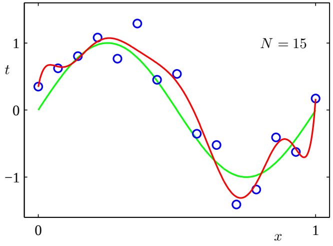


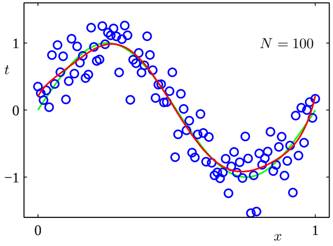


But there isn't always going to be more data…

# Overfitting

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



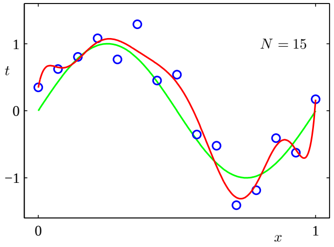


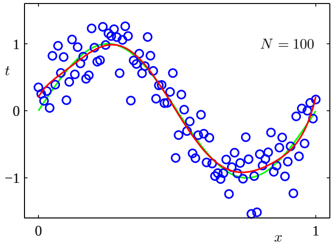
But there isn't always going to be more data…

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]

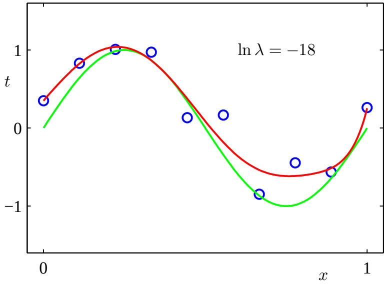
Let's keep that plan, but add an extra term to control the weights

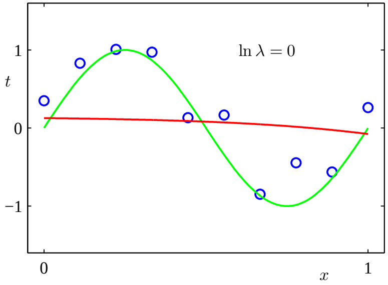
$$E_{\lambda} = \frac{1}{2}\sum_{n}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_{m}^{M} w_m^2$$

and a hyperparameter $\lambda$ to say how seriously we should take it as an error component

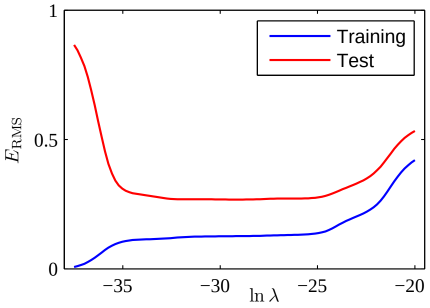
[1]The 1/2 is there to hint that this is the log likelihood for a Normal distribution (with constant error variance, so it doesn't matter to $E$)

## 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 $\lambda$ allows more of the model class's inherent flexibility to show

## Choosing hyperparameter values

We can't fit $\lambda$ by minimising the sum of squares

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

## Choosing hyperparameter values

We can't fit $\lambda$ by minimising the sum of squares

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

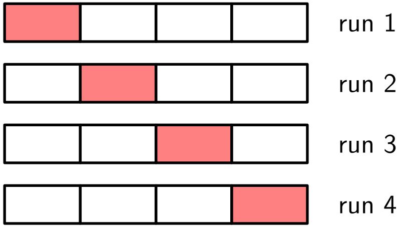
One reliable option is crossvalidation (CV)

→ 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



## General theory: Bias-Variance

One important question we can ask is about the expected value of $E$, averaged over *all possible data sets* coming from the same mechanisms

First, let's give $\mathrm{E}[t \mid x]$ (the *real* regression function) a name

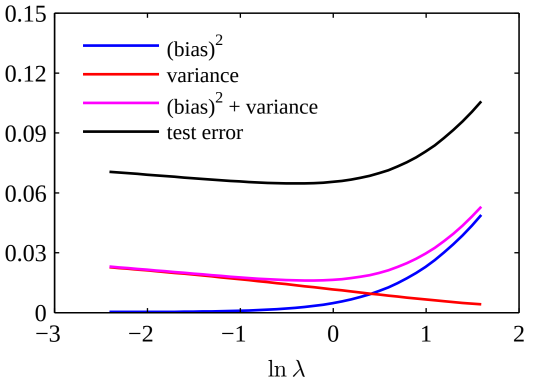
$$h(x) = \mathrm{E}[t \mid x]$$

and define $\mathrm{E}_D$ as an average over all possible data sets

Then (Bishop, sec. 1.5.5 and 3.2 for a derivation) we can decompose the expected error into

$$\begin{array}{rl} \text{bias}^2 & \int \mathrm{E}_D[(y(x,\mathrm{w}) - h(x))^2]\, p(x)\mathrm{d}x \\ \text{variance} & \int \mathrm{E}_D[(y(x,\mathrm{w}) - \mathrm{E}_D[y(x,\mathrm{w})])^2]\, p(x)\mathrm{d}x \\ \text{noise} & \int (h(x) - t)^2\, p(x,t)\, \mathrm{d}x\, \mathrm{d}t \end{array}$$

## Bias and Variance



Error is unavoidable but bias and variance trade off

## Model bias

Informally, the *bias* of a model class is the set of functions that a model most naturally learns

→ Linear models (M=1 above): can learn straight lines

→ Quadratic models (M=2 above): *can* learn straight lines but also smooth curves

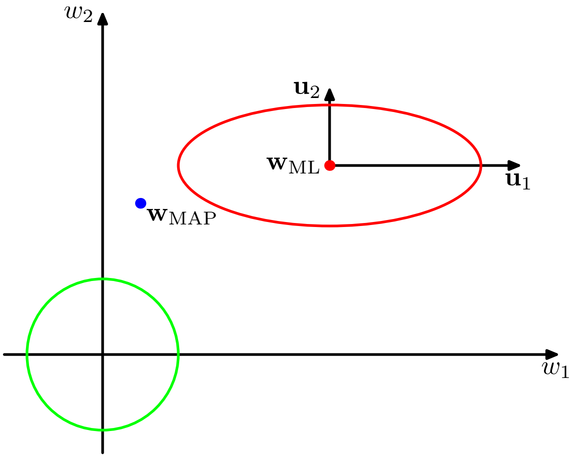
→ etc.

We can get different sorts of bias by changing the whole model class

→ we'll see an example of later with trees

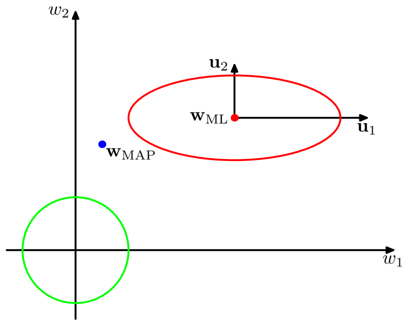
Regularization also offers us some interesting and different forms

## Model bias



$w_2$
$\mathbf{u}_2$
$\mathbf{w}_{\mathrm{ML}}$
$\mathbf{u}_1$
$\mathbf{w}_{\mathrm{MAP}}$
$w_1$

## Model bias

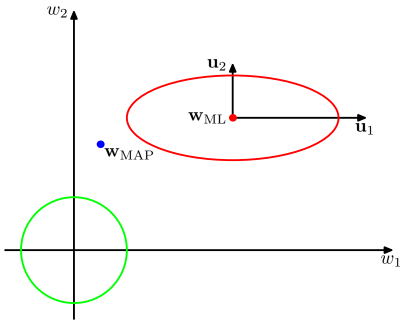


→ $\mathbf{w}_{\mathrm{ML}}$ minimizes $E_{\lambda=0} = E_{\mathrm{OLS}}$

→ The origin minimizes $E_{\lambda=\infty}$

→ $\mathbf{w}_{\mathrm{MAP}}$ minimizes $E_{\lambda}$ when we set $\lambda$ sensibly to balance the two parts of the error function

## Model bias



- → This bias *shrinks* all the weights, some more than others
- → It is sometimes helpful to define an *effective* number of parameters, which is less than M, and possibly fractional

## A different model bias

If we change the regularization term just a little (note the $q$)

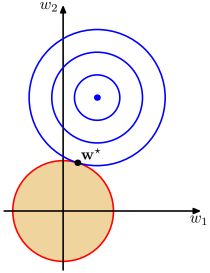
$$E_\lambda = \frac{1}{2}\sum_{n}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_{m}^{M}|w_m|^q$$

## A different model bias

If we change the regularization term just a little (note the $q$)

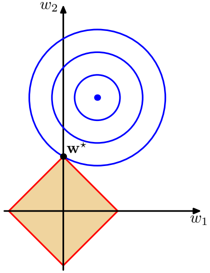
$$E_\lambda = \frac{1}{2}\sum_{n}^{N}(y(x_n,\mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\sum_{m}^{M}|w_m|^q$$

$q = 2$



(L2 regularization a.k.a. ‘ridge regression’)

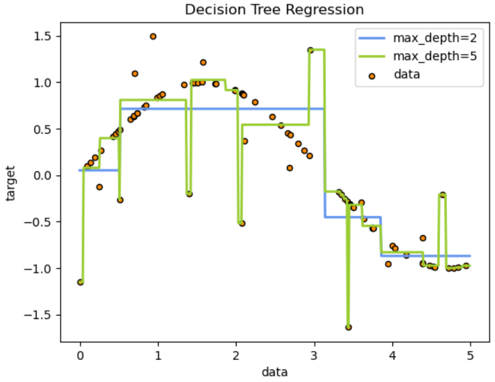
$q = 1$



(L1 regularization, a.k.a. ‘the lasso’)

## Totally different bias

Alternatively, we can change the model class 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, 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)

→ bootstrapping the dataset

→ Fitting trees to each bootstrap sample

→ Averaging the resulting predictions

or variations on that theme (Cutler et al., 2012, e.g. Random forests)

Like cross-validation, this removes variance but does not much affect bias

## Bias as a good thing

Clearly regularization generates bias. Seems like a bad thing…

But it's necessary

- → 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

## Bias as a good thing

Clearly regularization generates bias. Seems like a bad thing…

But it's necessary

- → 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

## Bias as a good thing

Clearly regularization generates bias. Seems like a bad thing…

But it's necessary

- → 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

- → Slows convergence

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

## Bias and variance in ML

ML insight:

→ It's better to working with a universal function approximator, and figure out how to regularize it, than to work with a model that can't represent much of anything and hope

→ Most of the ML methods we'll work with are universal approximators

→ Linear regression…definitely not.

## Bias and variance in ML

ML insight:

→ It's better to working with a universal function approximator, and figure out how to regularize it, than to work with a model that can't represent much of anything and hope

→ Most of the ML methods we'll work with are universal approximators

→ Linear regression…definitely not.

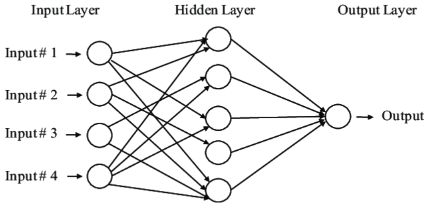
But wait, how did all that regularization business turn $y(x, \mathbf{w})$ into a universal approxiator?

→ It didn't. We just didn't say much about $y(x, \mathbf{w})$ and drew it like it was in a linear regression context

In real applications, $y(x, \mathbf{w})$ is not even polynomial regression

→ It's *kernel regression*, or *basis function* regression, *neural network*, *random forest*, etc.

## 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 $\phi_j$ 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

## ML models

We've some really flexible models with interesting different types of bias (smooth, piecewise linear) and styles of regularization (L1, L2, depth constraints)

Let's go right back to the beginning

- → Good old multiple linear regression

## ML models

We've some really flexible models with interesting different types of bias (smooth, piecewise linear) and styles of regularization (L1, L2, depth constraints)

Let's go right back to the beginning

→ Good old multiple linear regression

## Extending regression

Frisch and Waugh, back in 1933 in Econometrica showed, and Lowell generalized in 1963 the following useful fact about regression (Lovell, 2008, has a short accessible proof).

Consider three models

$$Y = \beta_0 + X\beta_X + Z_1\beta_{Z1}\ldots Z_K\beta_{ZK} + \epsilon \quad \text{(Big Model)}$$

$$X = \beta_0^X + Z_1\beta_{Z1}^X\ldots Z_K\beta_{ZK}^X + \epsilon^X \quad \text{(X Model)}$$

$$Y = \beta_0^Y + Z_1\beta_{Z1}^Y\ldots Z_K\beta_{ZK}^Y + \epsilon^Y \quad \text{(Y Model)}$$

and also this one, made out of residuals from the Y and X models

$$(Y - \hat{Y}) = \beta_0^{\text{FWL}} + (X - \hat{X})\beta^{\text{FWL}} + \epsilon^{\text{FWL}}$$

then

$$\beta_X = \beta^{\text{FWL}}$$

# Extending regression

What if $K$ were really big and we had no real idea about all those $Z$s?

- Lots of confounders
- Unknown possibly non-linear relationships
- Target effect is $\beta_X$, which we'll assume here *is* linear

## Extending regression

What if $K$ were really big and we had no real idea about all those $Z$s?

- → Lots of confounders
- → Unknown possibly non-linear relationships
- → Target effect is $\beta_X$, which we'll assume here *is* linear

Double/debiased Machine Learning (Chernozhukov et al., 2018)

Intuition: Do FWL but with *fancier* X and Y models

$$X = m(Z_1, \dots Z_K) + \epsilon^X \qquad \text{(Fancy X Model)}$$

$$Y = g(Z_1, \dots Z_K) + \epsilon^Y \qquad \text{(Fancy Y Model)}$$

## Double ML

Chernozhukov et al. show that

- → Just learning a fancier big model is a bad idea: the effect of X gets lost, the fancy model might throw it away, bias, etc.

However using fancy $m$ and fancy $g$ comes with problems:

- → Overfitting, due to flexibility of the model class
- → Bias, due regularization to combat overfitting
- → Slow convergence. we're used to $n^{1/2}$, but fancy models tend to go go $n^{1/4}$

They use a mixture of

- → cross-fitting: like cross validation but for $\beta_X$ estimation
- → cunning orthogonal score functions

To get fancy models that converge (mostly) as if they were simple ones. Cool

## References

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

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

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W. & Robins, J. (2018). ‘Double/debiased machine learning for treatment and structural parameters’. *The Econometrics Journal*, *21*(1), C1–C68.

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.

Lovell, M. C. (2008). ‘A simple proof of the fwl theorem’. *The Journal of Economic Education*, *39*(1), 88–91
_eprint: https://doi.org/10.3200/JECE.39.1.88-91.

## References

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