Ensemble Methods

Oxford Spring School in Advanced Research Methods, 2021

Dr Thomas Robinson, Durham University

Day 5/5

Recap

Over past three sessions:

- Explored three different broad categories of ML estimation:
  1. Extensions to OLS
  2. Tree-based methods
  3. Neural networks

Each attempt to make good out-of-sample in different ways, but share some foundational concepts:

- Bias-variance trade off
- Regularisation
- Hyperparameter tuning

How do we know which model to use and when?

- What if different models yield different predictions?

Today's session

Introduce the concept of ensemble learning:

- Combine multiple estimators to *improve* the overall predictive performance
- Different estimators might capture different components of covariance
- Can we combine the methods into a "super-learner"?

Rest of the session:

1. Basic ensemble methods
2. Superlearner theory
3. Causal inference and ML

ML as learning

To go back to the fundamentals of ML:

- Different models represent different ways to *learn* the relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$
  - By projecting points onto a lower dimensional surface (OLS)
  - By subsetting data into ever more homogenous groups (trees)
  - By approximating complex functional forms (neural nets?)

We can think about two levels of optimisation:

- $\arg\min_{\lambda,\theta} |f(\boldsymbol{X}) - \hat{g}_{\lambda,\theta}(\boldsymbol{X})|$
  - Solved algorithmically and via tools like cross-validation
- $\arg\min_{g} |f(\boldsymbol{X}) - \hat{g}_{\lambda,\theta}(\boldsymbol{X})|$
  - Impossible to solve in most cases

Generic meta-prediction problem

How can we optimise our choice of "learner" in order to best approximate the true data generating process?

- Multiple strategies to choose from
- Multiple specific estimators within strategies

Naive approach:

- Run multiple models, choose the one with lowest training error
- *Why might this not be the best strategy?*

Combining models

Choosing a *single* best model may still be suboptimal:

- Different estimating strategies may be better at modeling different parts of the outcome

The alternative is to combine models:

- We can average models together
- We can adapt models iteratively
- We can "weight" and sum models

Do any of these sound familiar?

Overarching intuition is to leverage the different capacities of an **ensemble** of models in order to improve prediction performance

Ensemble methods

Three basic approaches to combining ML models

Bagging

- "Bootstrap aggregation", random samples with replacement as training data
- Typically used to reduce variance

Boosting

- Sequential weak-learner models, data iteratively weighted by misclassification
- Typically used to reduce bias

Stacking

- Run different estimation strategies on the *same* training data, then combine estimates
- Removes model selection issues and improves predictive performance

Bagging

We've already briefly covered bagging – it helps convert CART models into a random forest

- Bagging = "**Bootstrap aggregation**"
- We train B *independent* models of the same type
  - If $\boldsymbol{X}$ were the same across all B models, and assuming a deterministic estimator, then there would be no variance in the predictions
  - Therefore we induce variation by bootstrapping the data
    - Let n be the number of rows in $\boldsymbol{X}$
    - Randomly sample n rows of the data *with replacement*
    - Yields B datasets of the same size as $\boldsymbol{X}$ (with the same variables)
    - Rows of $\boldsymbol{X}$ can repeat within any given sample

Bagging

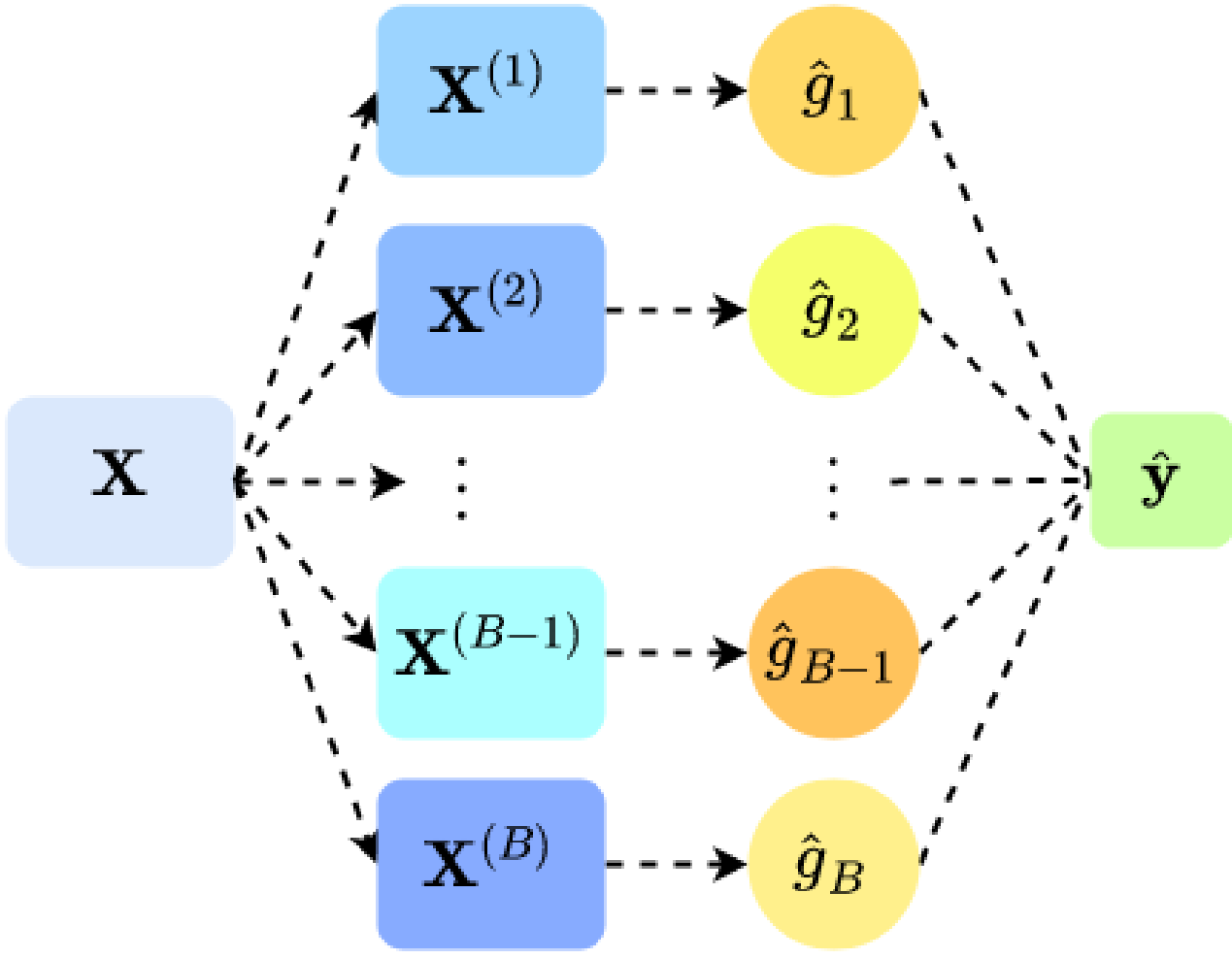
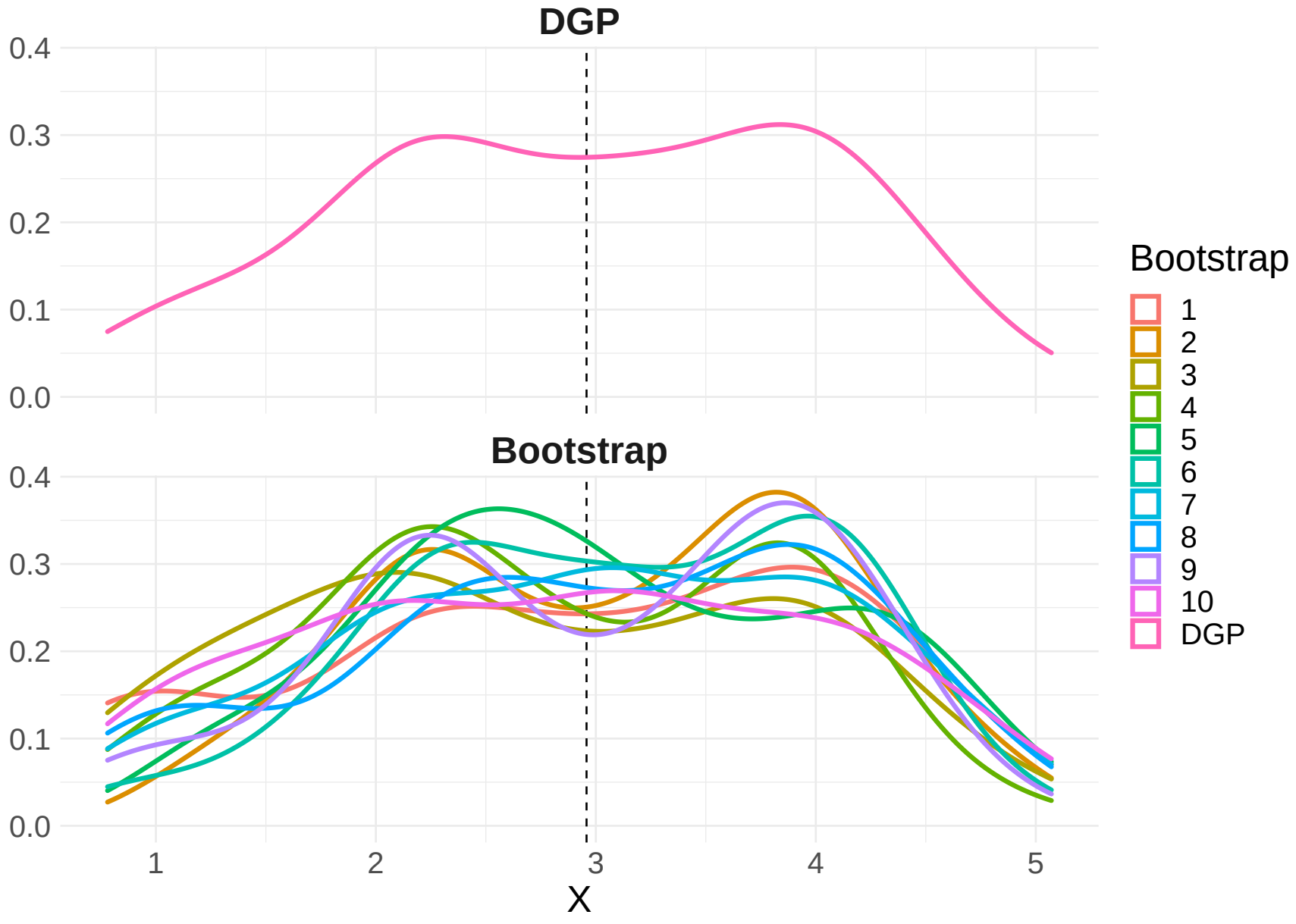


Figure 1: Schematic of a bagged ensemble

Visualising the bootstrap



Bagging details

If $\boldsymbol{X}$ is the training data:

- Let $\mathcal{B} = \{\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(B)}\}$ be a set of randomly resampled datasets, each the length of $\boldsymbol{X}$
- Let g be an estimating function
- $\boldsymbol{X}'$ be the test data

Then:

- For each dataset $\boldsymbol{X}^{(b)} \in \mathcal{B}$, estimate $\hat{g}_b = g(\boldsymbol{X}^{(b)})$
- Then, pass the test data through each separate model
  - $\hat{\boldsymbol{y}}_b = \hat{g}_b(\boldsymbol{X}')$
- $\hat{y}_i = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_{ib}$

Why is bagging useful?

Bagging is typically used with **high variance** estimators:

- I.e. the modelling strategy likely overfits the data
- Yielding predictions that are very sensitive to the input data
- By bootstrapping $\hat{X}$ we can smooth out this sensitivity
- And yield predictions that exhibit less variance

Categorical outcomes

The above strategy works very well for continuous data, but what about categories?

Two methods:

1. **Hard vote**

- Each model acts as a "voter", choosing the class with the highest predicted probability *within* that model
- $\hat{y}_i$ gets assigned the class with the most "votes" across the B models

2. **Soft vote**

- For every class, average the predicted probabilities from the B models
- $\hat{y}_i$ gets assigned the class with the highest *average* predicted probability

Parallelisation

Particularly convenient aspect of bagging is that it can be **parallelised**:

- Since each model is independent, the estimation is entirely separate
- Thus, separate compute units can be used on each model
- Or, more realistically:
  - If we have C cores on a PC, each core can train $\frac{B}{C}$ models
  - In other words, C models can be trained at once
  - If the actual computation is not too taxing, then this can dramatically improve speed of training

Parallelisation is used across ML:

- Some calculations are very simple but there are simply thousands of them
- Can shift these away from CPU to GPU, as used in neural network training

Boosting

Bagging helps address the problem of high variance methods

- What about high *bias* methods?
- We might have modeling strategies that exhibit low variance...
  - E.g. (very) shallow CART trees
- ... but are poor predictors of some observations

Boosting operates on the principle of training models sequentially

- The next model g_{m+1} attempts to address parts of $\boldsymbol{y}$ that g_m predicted less well
  - I.e. correcting previous mistakes
- One common approach is to boost using **adaptive boosting**
  - Popular classification ensemble method

Adaptive boosting

Under adaptive boosting, the ensemble model is a weighted sum of "weak learners"

$$S_L(\boldsymbol{X}) = \sum_{l=1}^{L} c_l \times w_l(\boldsymbol{X}),$$

where:

- L indexes a series of weak learners
- c_l is a coefficient corresponding to the lth weak learner w_l

To optimise $\boldsymbol{c}$ and $\boldsymbol{w}$, we add the models sequentially to S_L

The optimisation function at each iteration is defined as:

$$\arg\min_{c_l, w_l} \sum_{i=1}^{N} e(y_i, s_{l-1}(x_i) + c_l \times w_l(x_i))$$

where e is the prediction error function

Boosting

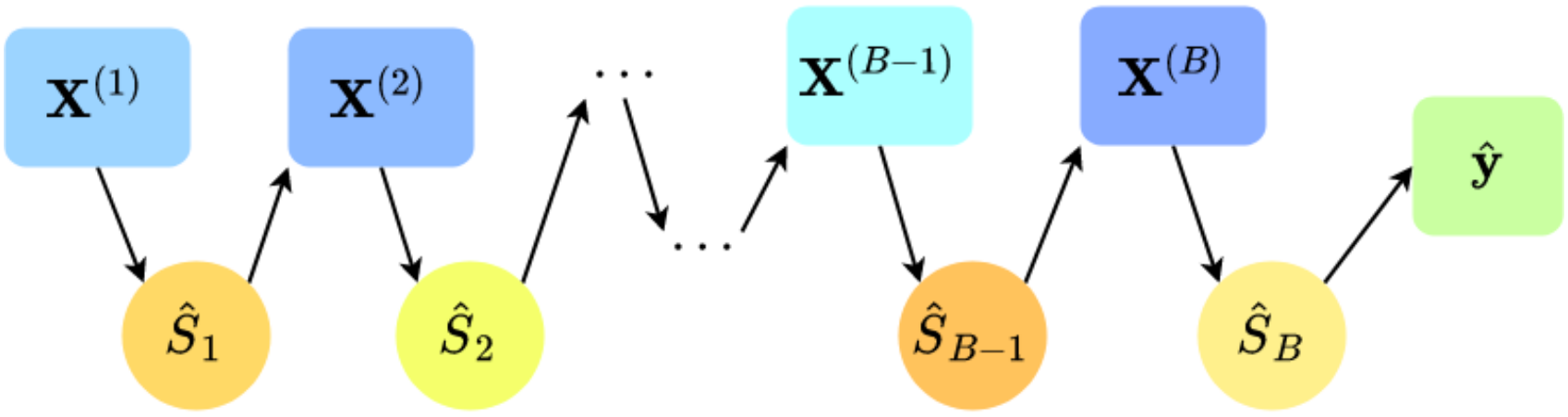


Figure 2: Schematic of a boosted ensemble

Adaptive boosting classifier algorithm

Let $\boldsymbol{\psi}^{(1)} = \mathbf{1}$ (the initial weights for each observation in $\boldsymbol{X}$)

For l in $1 : L$

1. Find w_l that minimises the following loss

$$\epsilon_l = \frac{\sum_{i=1}^{N} \psi_i^{(l)} \times \mathbb{I}(y_i \neq w_l(\boldsymbol{x}_i))}{\sum_i \psi_i^{(l)}}$$

2. Given $\hat{w}_l$, compute its coefficient

$$c_l = \ln \frac{1 - \epsilon_l}{\epsilon_l}$$

3. Update each observation weight, such that

$$\psi_i^{(l+1)} = \psi_i^{(l)} \times e^{c_l \mathbb{I}(w_l(\boldsymbol{x}_i) \neq y_i)}$$

Then, $\hat{S}_L(\boldsymbol{X}) = \text{sign}(\sum_{l=1}^{L} c_l \hat{w}_l(\boldsymbol{X}))$

Parallelisation?

We cannot boost a model in parallel

- Each subsequent model is fit to address the deficiencies of the pre-existing model
- Thus S_l is dependent on the learner S_{l-1}

Potentially inflates training times

- To boost a model with 1000 iterations is going to take $1000 \times c$ to learn
- Offset if our low-bias individual models are simple
- Suppose we have 4 cores available
  - Complex bagged model takes $10c$ to run, and we have 1000 iterations
    - Total time $= \frac{1000}{4} \times 10c = 2500C$
  - Simple model takes $1c$ and we run 1000 models
    - Total time $= 1000c$

Stacking

Stacking also works by combining models, but here allows the *type* of model to vary.

The general intuition is:

- Individual learning strategies may be "weak"
- Any one model is unlikely to be optimal
- Leverage different predictive capacities of M strategies to improve prediction

This is a meta-regression strategy:

- We assume a model H such that $\boldsymbol{y} = H(\boldsymbol{X})$
- The constituent parameters of the super-regression are the outputs from individual-learning strategies

Stacking

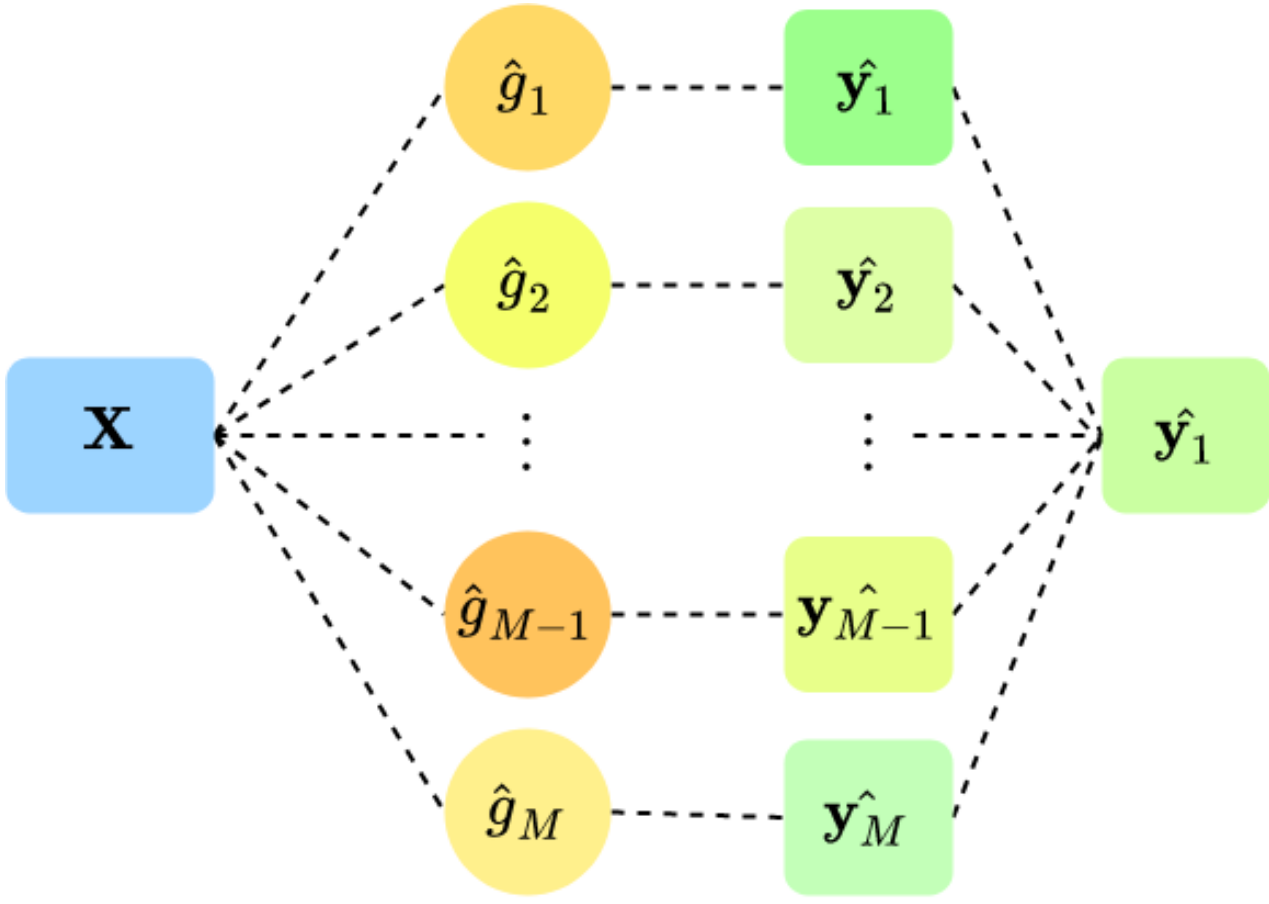


Figure 3: Schematic of a stacked ensemble

Estimation steps

Four steps to generating stacked ensemble predictor:

1. Isolate M learning strategies
2. Train M strategies using k-fold validation
3. Estimate parameters of meta-regression (the "super-learner")
4. Predict new data using trained ensemble

1. Isolate learning strategies

Can use any modelling strategies:

- Just need to be able to recover individual $\hat{y}$ predictions
- In general, better to include more models

Model choices:

- OLS
- LASSO
- CART
- Random Forest
- BART
- Neural networks, etc.

Let $\mathcal{M}$ be the set of chosen strategies (the "library"):

- Let g_m be a learning strategy $m \in \mathcal{M}$
- $\hat{g}_m$ is the trained model on training data $\boldsymbol{X}$
- $\hat{y} = \hat{g}_m(\boldsymbol{X})$ is the predicted outcomes of the training data

2. Cross-fold validation for *each* strategy

Let $\mathcal{V}$ be a set of random folds $\{1, \ldots, V\}$, where

- $\boldsymbol{X}^{(v)}$ is the training data from the vth fold
- $\boldsymbol{X}^{(-v)}$ is the training data excluding the vth fold

For $v \in \mathcal{V}$:

- Train each strategy g_m on $\boldsymbol{X}^{(-v)}$
- Generate predicted values for the vth fold
  - I.e. $\hat{\boldsymbol{y}}_m^{(v)} = \hat{g}_m(\boldsymbol{X}^{(v)})$

Concatenate the predictions for each m

- $\hat{\boldsymbol{y}}_m = \{\hat{\boldsymbol{y}}_m^{(1)}, ..., \hat{\boldsymbol{y}}_m^{(V)}\}$

3. Estimate super learner weights

With M vectors of predictions for $\boldsymbol{y}$:

- Create a $n \times m$ matrix by stacking predictions from each model

Then we estimate an OLS model:

$$y_i = \sum_{m=1}^{M} w_m \hat{y}_{im} + \epsilon_i.$$

So that w_m are weights, we impose two further restrictions on $\boldsymbol{w}$:

- $w_m \geq 0$
- $\sum_m w_m = 1$

Denote the resultant trained model $\hat{H}$

4. Predict new data

From stages 1-3 we have:

- A library of trained models on $\boldsymbol{X}$
- Estimated weights for each model in $\mathcal{M}$

To make predictions on *new* test data $\boldsymbol{X'}$ we:

1. Pass the test data through each model in $\mathcal{M}$
   - I.e. $\hat{\boldsymbol{y'}}_m = \hat{g}_m(\boldsymbol{X'})$, where $\hat{g}_m$ has been pre-trained on $\boldsymbol{X}$
2. Pass the resultant predictions through $\hat{H}$
   - I.e. $\hat{\boldsymbol{y'}} = \hat{H}(\boldsymbol{X'}) = \hat{\alpha} + \sum_{m=1}^{M} \hat{w}_m \hat{\boldsymbol{y'}}_m$

Oracle performance

This "super-learner" strategy is particularly effective because it is an **oracle learner**:

> *The cross-validated super-learner model "perform asymptotically exactly as well... as the best possible choice for the given data set among the family of weighted combinations of the estimators"*

This means the super-learner really is "super":

> *"Super learner will asymptotically outperform any of its competitors, even if the set of competitors is allowed to grow polynomial in sample size"*
>
> – Polley and Van der Laan 2010

Parallelisation

Nothing precludes training the M models simultaneously:

- Modelling across strategies is parallel
- Devoting fewer cores to each method may yield slower aggregate performance if individual models themselves rely on parallelism (i.e. random forest)

The weighting model can only be estimated once all models are trained

- OLS regression should be trivially quick

Application to heterogeneous effects prediction

Suppose we have 8 observations of an outcome, treatment assignment and two covariates:

y	d	Gender	Education
12	1	Female	High
13	1	Female	Low
5	0	Female	High
6	0	Female	Low
7	1	Male	High
8	1	Male	Low
7	0	Male	High
6	0	Male	Low

Table 1: Observed

y	d	Gender	Education
?	0	Female	High
?	0	Female	Low
?	1	Female	High
?	1	Female	Low
?	0	Male	High
?	0	Male	Low
?	1	Male	High
?	1	Male	Low

Table 2: Unobserved counterfactual

$$\text{ATE}_{\text{Observed}} = 10 - 6 = 4$$

The ATE may mask considerable heterogeneity

Conditional Average Treatment Effect

We can break down our ATE into conditional estimates:

$$\text{CATE} = \mathbb{E}[Y|d = 1, X = a] - \mathbb{E}[Y|d = 0, X = a]$$

In particular, we can think about estimating the individual level effect, i.e.

$$\text{ITE} = [Y_i|d = 1] - [Y_i|d = 0]$$

- This is not an average
- Conventionally impossible to estimate given **fundamental problem of causal inference**

Prediction problem:

- For those treated (control) units, what would they have done under control (treatment)?

Ensemble estimation strategy

1. Train a stacked ensemble estimator on the observed data ($\boldsymbol{X}$), using a library of methods:
   - BART
   - Random forest
   - Shallow CART
   - Neural network
   - etc.
   - Yields $\hat{H}$ – our ensemble estimator
2. Predict $\hat{H}(\boldsymbol{X}^{(\neg d)})$
   - Counterfactual data where if $d_i = 1, \neg d_i = 0$
   - All other variables are identical
3. Calculate ITE

Estimating the CATE - generate two test matrices

- Predictions are made using two matrices
- Second matrix is the test dataset with predicted outcomee
- Matrices are identical except treatment assignment is reversed in second matrix

$d_{\text{Obs.}}$	**Gender**	**Education**	$y_{i,d}$
1	*Female*	*High*	14
1	*Female*	*Low*	12
0	*Female*	*High*	4
0	*Female*	*Low*	6
1	*Male*	*High*	7
1	*Male*	*Low*	7
0	*Male*	*High*	8
0	*Male*	*Low*	6

$d_{\text{Counter.}}$	**Gender**	**Education**	$\hat{y}_{i,d}$
0	*Female*	*High*	7
0	*Female*	*Low*	7
1	*Female*	*High*	12
1	*Female*	*Low*	13
0	*Male*	*High*	8
0	*Male*	*Low*	6
1	*Male*	*High*	8
1	*Male*	*Low*	6

Estimating the CATE - rearrange matrices

- Matrices can be rearranged such that all observations in matrix 1 are $d = 1$ and *vice versa*
- Covariate information is constant across both matrices

$d_{\text{Obs.}}$	**Gender**	**Education**	$y_i^{(d=1)}$
1	*Female*	*High*	14
1	*Female*	*Low*	12
1	*Female*	*High*	12
1	*Female*	*Low*	13
1	*Male*	*High*	7
1	*Male*	*Low*	7
1	*Male*	*High*	8
1	*Male*	*Low*	6

$d_{\text{Counter.}}$	**Gender**	**Education**	$y_i^{(d=0)}$
0	*Female*	*High*	7
0	*Female*	*Low*	7
0	*Female*	*High*	4
0	*Female*	*Low*	6
0	*Male*	*High*	8
0	*Male*	*Low*	6
0	*Male*	*High*	8
0	*Male*	*Low*	6

Estimating the CATE - recover CATE

- CATE $= y_i^{(d=1)} - y_i^{(d=0)}$
- To check for treatment heterogeneity, append covariate information since this is constant across two matrices[1]

$$\begin{pmatrix} \widehat{\mathbf{y}}_{\mathbf{i},\mathbf{d}=\mathbf{1}} \\ 14 \\ 12 \\ 12 \\ 13 \\ 7 \\ 7 \\ 6 \\ 7 \end{pmatrix} - \begin{pmatrix} \widehat{\mathbf{y}}_{\mathbf{i},\mathbf{d}=\mathbf{0}} \\ 7 \\ 7 \\ 4 \\ 6 \\ 8 \\ 6 \\ 8 \\ 6 \end{pmatrix} = \left(\begin{array}{c|cc} \textbf{CATE} & \textbf{Gender} & \textbf{Education} \\ 7 & \textit{Female} & \textit{High} \\ 5 & \textit{Female} & \textit{Low} \\ 8 & \textit{Female} & \textit{High} \\ 7 & \textit{Female} & \textit{Low} \\ -1 & \textit{Male} & \textit{High} \\ 1 & \textit{Male} & \textit{Low} \\ -2 & \textit{Male} & \textit{High} \\ 1 & \textit{Male} & \textit{Low} \end{array}\right)$$

[1] NB: all observations are predicted from posterior draws; red numbers indicate predictions using counterfactual treatment assignment

ML is not a panacea

ML: what can go wrong?

ML is really powerful:

- Individual models can be highly predictive
- Aggregate of M modeling strategies can be even better (and no worse!)
- Clear relevance to prediction problems

But we are not absolved of inference issues

- ML involves *biased* estimators
- By definition, these will not centre on *true* parameter estimate

Is this problematic?

- OLS may be no better (Gauss Markov assumptions are strong)
- But if GM does hold, then we *would* recover unbiased estimate

Incorrect interpretation

"I ran a LASSO model to identify the most important predictors from a high-dimensional dataset. For every unit increase in GDP, perceived levels of democracy increase by 20%!"

1. What's good about this statement?
2. What's bad about this statement?

Neglecting design

> *"It's impossible to directly observe levels of trust for historical periods. Therefore we take a library of portrait paintings from across time, and use a neural network-based facial recognition strategy to predict trustworthiness. When we regress a measure of democracy over time with this measure, we find that increases in trust lead to increases in democratic values!"*

1. What's good about this statement?
2. What's bad about this statement?