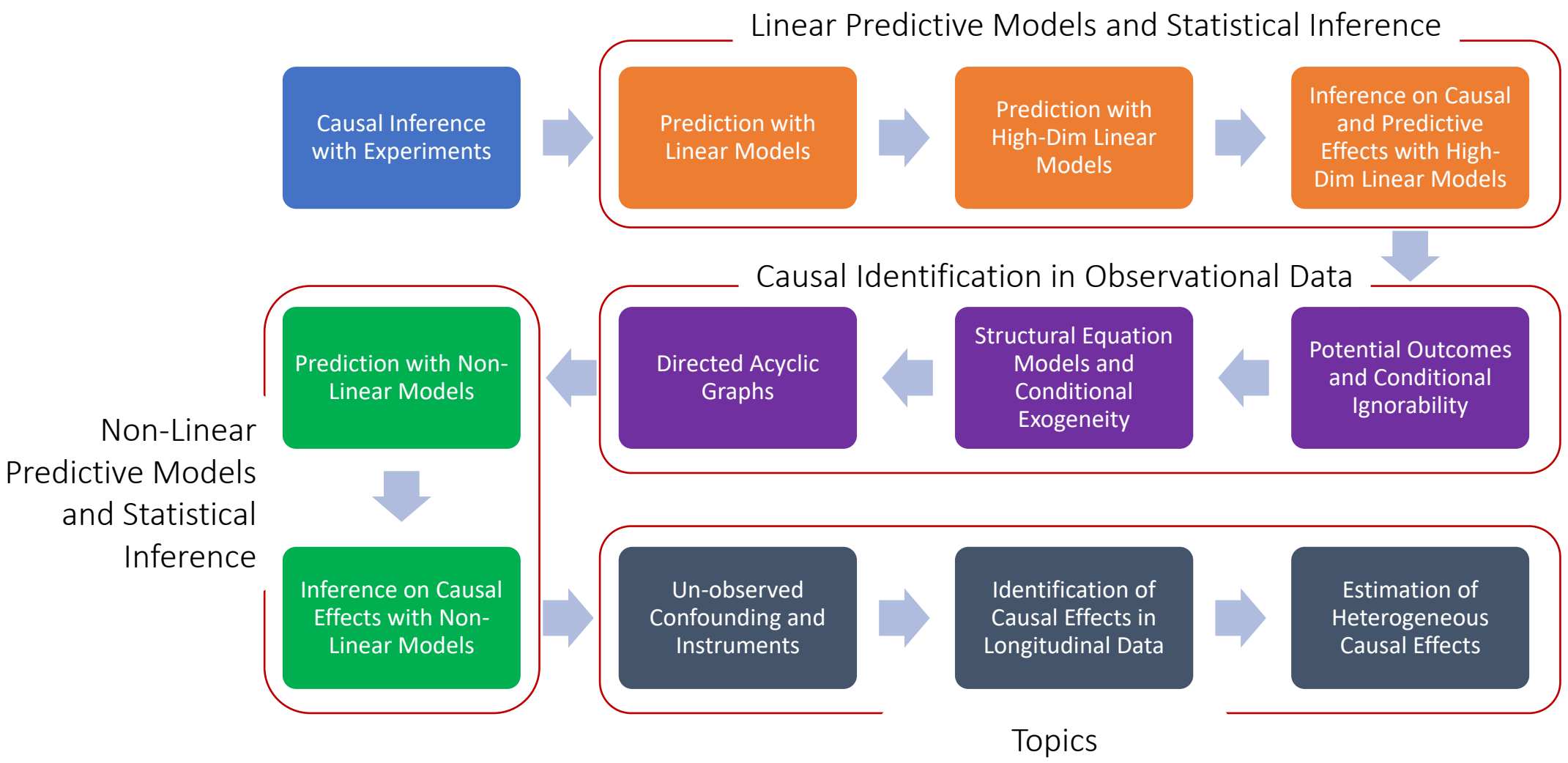
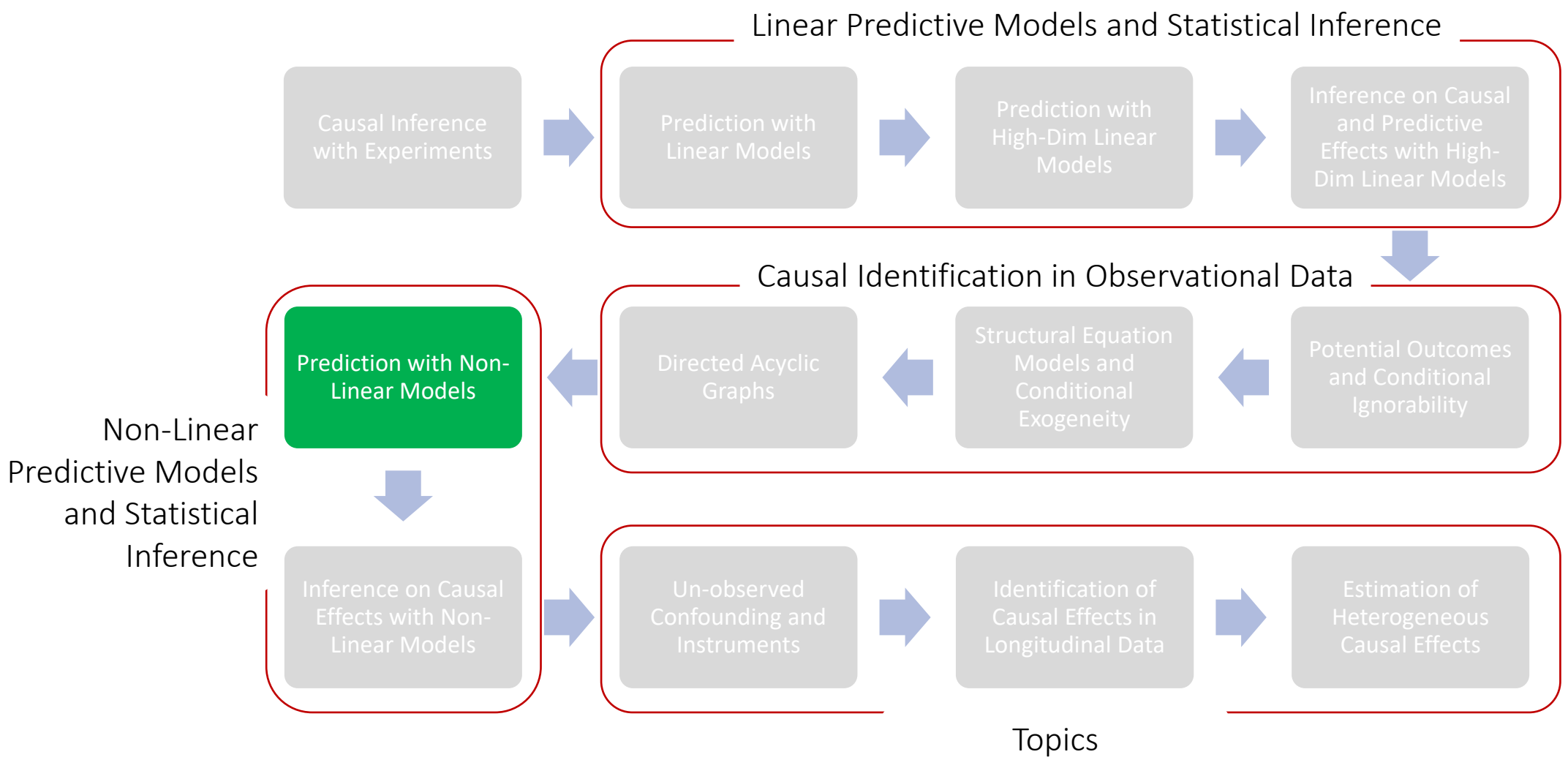
MS&E 228: Modern Non-Linear Prediction

Vasilis Syrgkanis

MS&E, Stanford


Linear Predictive Models and Statistical Inference
Causal Inference with Experiments
Prediction with Linear Models
Prediction with High-Dim Linear Models
Inference on Causal and Predictive Effects with High-Dim Linear Models
Causal Identification in Observational Data
Potential Outcomes and Conditional Ignorability
Structural Equation Models and Conditional Exogeneity
Directed Acyclic Graphs
Non-Linear Predictive Models and Statistical Inference
Prediction with Non-Linear Models
Inference on Causal Effects with Non-Linear Models
Un-observed Confounding and Instruments
Identification of Causal Effects in Longitudinal Data
Estimation of Heterogeneous Causal Effects
Topics

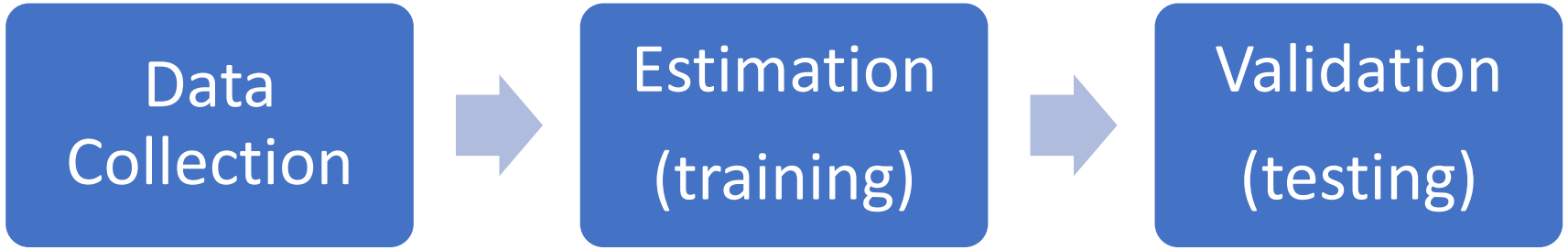

Linear Predictive Models and Statistical Inference
Causal Inference with Experiments
Prediction with Linear Models
Prediction with High-Dim Linear Models
Inference on Causal and Predictive Effects with High-Dim Linear Models
Causal Identification in Observational Data
Potential Outcomes and Conditional Ignorability
Structural Equation Models and Conditional Exogeneity
Directed Acyclic Graphs
Prediction with Non-Linear Models
Non-Linear Predictive Models and Statistical Inference
Inference on Causal Effects with Non-Linear Models
Un-observed Confounding and Instruments
Identification of Causal Effects in Longitudinal Data
Estimation of Heterogeneous Causal Effects
Topics

Recap

The Predictive Modelling Pipeline

Theory

Guarantee that for the sample size we have the resulting solution will be close to the truth under “inductive bias” assumptions on the truly best predictive model (e.g. sparsity)



Practice

e.g. Fit a lasso, OLS, Ridge, ElasticNet model on half of the data

e.g. Calculate RMSE and R-square on the other half and measure performance

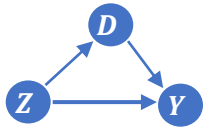
The Causal Modelling Pipeline

Theory

Formal Mathematical Frameworks for Expressing Assumptions

$Y(d) \perp\!\!\!\perp D \mid Z$ — Conditional Ignorability

$Y \coloneqq aD + \beta' Z + \epsilon_Y$
$D \coloneqq \gamma' Z + \epsilon_D$ — Linear SEMs



DAGs/ASEMs

Data Collection → Domain Assumption Elicitation → Estimation (training) → Validation (testing)

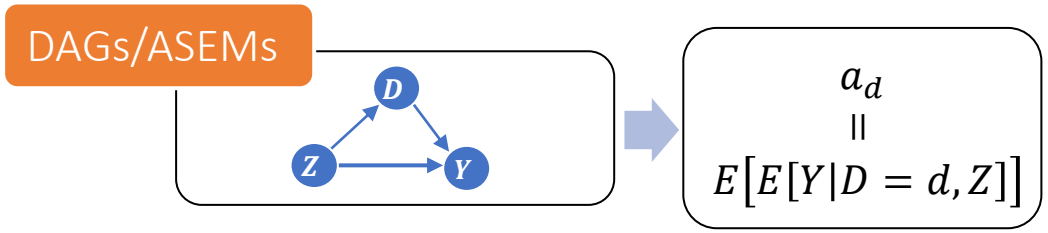
Practice

Ask domain expert on properties of data collection process

- **Conditional Ignorability.** "give me a set of variables Z such that potential outcomes are independent of treatment assignment given Z"
- **Structural Equations.** "describe the data (and unobserved data) via assignment equations and exogenous noise terms"
- **DAG.** "give me a graph that describes the potential influences of variables in (and not in) the data"

The Causal Modelling Pipeline

Theory



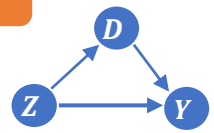
Practice

Given the domain assumptions, what are the predictive models I need to estimate in the data and how should I combine them to get a causal effect or a mean counterfactual response or a mean response under an intervention

The Causal Modelling Pipeline

Theory

DAGs/ASEMs



$$a_d$$
$$\parallel$$
$$E\left[E[Y|D=d,Z]\right]$$

Construct estimate $\hat{g}(d,z)$ of predictive model

$$E[\,Y \mid D=d, Z=z\,]$$

Estimate mean counterfactual

$$\hat{a}_d = E_n[\hat{g}(d,Z)]$$

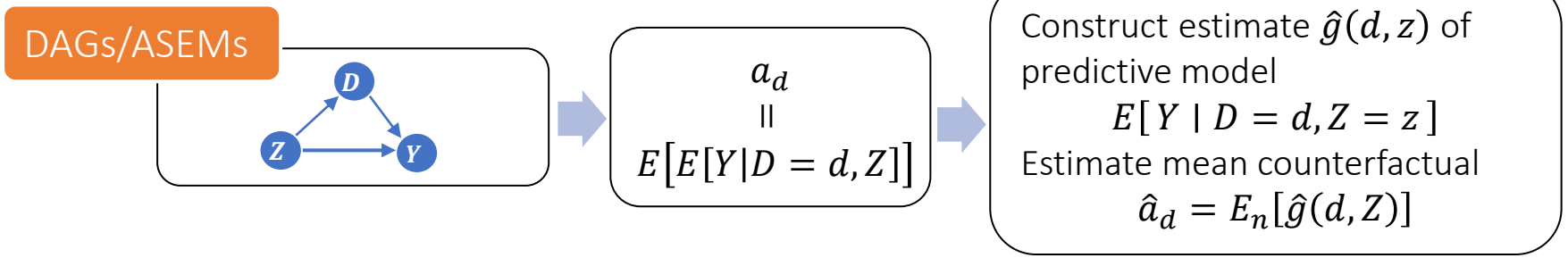
Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing)

Practice

So I need to estimate this bigger predictive model, predicting the outcome from the treatment D and the controls S, in some manner. Then average the predictive values over S fixing $d = 1$.

The Causal Modelling Pipeline

Theory



Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing)

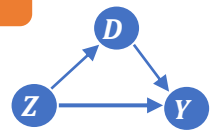
I can do normal cross-validation to evaluate the predictive model that I estimated

Practice

The Causal Modelling Pipeline

Theory

DAGs/ASEMs



a_d
=
$E[E[Y|D = d, Z]]$

Construct estimate $\hat{g}(d, z)$ of predictive model
$E[Y \mid D = d, Z = z]$
Estimate mean counterfactual
$\hat{a}_d = E_n[\hat{g}(d, Z)]$

$a_d \in \hat{a}_d \pm 2\hat{\sigma}$

Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing) → Inference (Confidence Intervals)

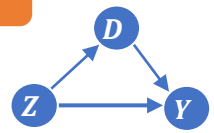
Practice

There is no way to validate my causal estimate $\hat{\alpha}_d$, given the data that I have! I cannot observe counterfactuals. The best alternative is to quantify uncertainty of my estimate to see how confident I should be in the result.

Sneak Peek in Future Lectures

Theory

DAGs/ASEMs



$$a_d = E[E[Y|D=d,Z]]$$

Construct estimate $\hat{g}(d,z)$ of predictive model

$$E[Y \mid D=d, Z=z]$$

Estimate mean counterfactual

$$\hat{a}_d = E_n[\hat{g}(d,Z)]$$

$$a_d \in \hat{a}_d \pm 2\hat{\sigma}$$

Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing) → Inference (Confidence Intervals) → Sensitivity Analysis

Practice

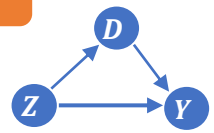
What if my domain assumptions were wrong? How confident should I be of the result? What if there is some remnant unobserved confounder? Can I at least give some bound on the correct answer?

$$a_d \in \hat{a}_d \pm (2\hat{\sigma} + \epsilon)$$

Sneak Peek in Future Lectures

Theory

DAGs/ASEMs



a_d
=
$E[E[Y|D = d, Z]]$

Construct estimate $\hat{g}(d, z)$ of predictive model
$E[Y \mid D = d, Z = z]$
Estimate mean counterfactual
$\hat{a}_d = E_n[\hat{g}(d, Z)]$

$a_d \in \hat{a}_d \pm 2\hat{\sigma}$

Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing) → Inference (Confidence Intervals) → Sensitivity Analysis

Practice

What decision or policy should I implement?
Should we treat? Who should be treated?

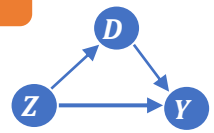
$a_d \in \hat{a}_d \pm (2\hat{\sigma} + \epsilon)$

Decision Policy

Practically Relevant Material we might not get to

Theory

DAGs/ASEMs



$$a_d = E[E[Y|D=d,Z]]$$

Construct estimate $\hat{g}(d,z)$ of predictive model

$$E[Y \mid D=d, Z=z]$$

Estimate mean counterfactual

$$\hat{a}_d = E_n[\hat{g}(d,Z)]$$

$$a_d \in \hat{a}_d \pm 2\hat{\sigma}$$

Practice

Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing) → Inference (Confidence Intervals) → Sensitivity Analysis → Decision Policy → Adaptive Experiments → Data Collection

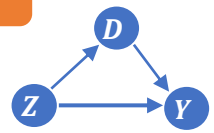
What RCT/experiment should I run next to validate my finding/policy and collect new data, validate/invalidate domain assumptions

$$a_d \in \hat{a}_d \pm (2\hat{\sigma} + \epsilon)$$

Today

Theory

DAGs/ASEMs



a_d
=
$E[E[Y|D=d,Z]]$

Construct estimate $\hat{g}(d,z)$ of predictive model
$E[Y \mid D=d, Z=z]$
Estimate mean counterfactual
$\hat{a}_d = E_n[\hat{g}(d,Z)]$

$a_d \in \hat{a}_d \pm 2\hat{\sigma}$

Practice

Data Collection → Domain Assumption Elicitation → Identification → Estimation (training) → Validation (testing) → Inference (Confidence Intervals) → Sensitivity Analysis → $a_d \in \hat{a}_d \pm (2\hat{\sigma} + \epsilon)$ → Decision Policy → Adaptive Experiments → Data Collection

So I need to estimate this **bigger predictive model, predicting the outcome from the treatment D and the controls S,** in some manner. Then average the predictive values over S fixing $d = 1$.

Goals for Today

- How do we estimate predictive models without linearity assumptions
- Overview of major mainstream machine learning methods
- Random Forests, Gradient Boosted Forests, Neural Networks
- Some theoretical guarantees and justification (similar to lasso)
- How to combine models (stacking)
- How to automate the process (automl)

Next lecture:

- Using ML as black-box feature engineering

Modern Non-Linear Predictive Models

Problem Statement

- Given n samples $(Z_1, Y_1), \ldots, (Z_n, Y_n)$ drawn iid from a distribution D
- Want an estimate $\hat{g}$ that approximates the Best Prediction

$$g \coloneqq \operatorname*{argmin}_{\tilde{g}} E\left[\left(Y - \tilde{g}(Z)\right)^2\right]$$

- Best Prediction rule is Conditional Expectation Function (CEF)

$$g(Z) = E[Y|Z]$$

- We want our estimate $\tilde{g}$ to be close to g in RMSE

$$\|\hat{g} - g\| = \sqrt{E_Z\left(\hat{g}(S) - g(Z)\right)^2} \to 0, \qquad \text{as } n \to \infty$$

Thus Far: Linear CEF

- If CEF is assumed linear with respect to known engineered features

$$E[\,Y \mid Z\,] = \beta' \phi(Z)$$

- Then the Best Prediction rule (CEF) coincides with the Best Linear Prediction rule (BLP)

- We can use OLS if $\phi(Z)$ is low-dimensional ($p \ll n$) or the multitude of approaches we learned if $\phi(Z)$ is high-dimensional (Lasso, ElasticNet, Ridge, Lava)

The Curse of Dimensionality

- What if we make no real assumption on $g(Z) \coloneqq E[Y|Z]$
- Suppose we only assume g is a smooth function
- Formal form of smoothness: g is β-smooth if it has uniformly bounded and continuous β-high order derivatives

- Classic non-parametric statistics [Stone'82]: provably best you can do

$$\|g - \hat{g}\| \approx n^{-\frac{\beta}{2\beta+p}}$$

The Curse of Dimensionality

- Say we have $p = 10$ variables (typical empirical application)
- Say we only assume uniformly bounded continuous derivative ($\beta = 1$)
- If we want an RMSE of 0.1 then we need

$$n^{-\frac{1}{12}} \approx 0.1 \Rightarrow n \approx 10^{12} = 1 \text{ trillion samples!}$$

- If we made a stronger assumption that second derivative is also uniformly bounded and continuous ($\beta = 2$)

$$n^{-\frac{2}{14}} \approx 0.1 \Rightarrow n \approx 10^{7} = 10 \text{ million samples}$$

Bypassing the Curse of Dimensionality

- Lasso scaled to $p \gg n$ by adapting to notions of "effective dimension" (e.g. s/n, with s is number of relevant variables)
- We need methods with similar behavior for non-linear models

- Many modern machine learning techniques achieve exactly that
- Their error scales with appropriate notions of "effective dimension"

- Some heuristically (open research), some with provable guarantees

Regression Trees

- Partition regressor space into a set of rectangles $R_1, \ldots, R_M$
- A simple model is then fit within each rectangle
- Simplest approach to fit a constant within each rectangle

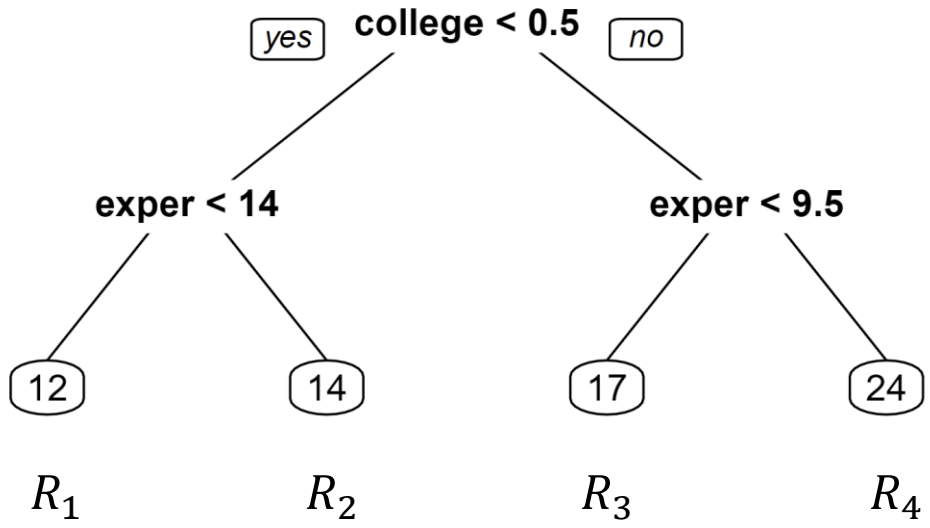
$$f(Z) = \sum_{j=1}^{M} \beta_j \, 1(Z \in R_j)$$

- The constants are fitted by minimizing the in-sample RMSE

$$\hat{\beta} \coloneqq \underset{\beta}{\operatorname{argmin}} \, E_n \left[\left(Y - \sum_{j} \beta_j \, 1(Z \in R_j) \right)^2 \right]$$

Regression Trees

- Rectangles are nicely represented as leafs in a binary decision tree
- Key difference with linear models: regions constructed based on the data (otherwise just a simple linear CEF)



Growing Regression Trees

- Regression trees are typically constructed in a “greedy” manner
- We start from all the data and find the regressor (e.g. college) and the splitting threshold (e.g. 0.5), that produces largest improvement of in-sample MSE

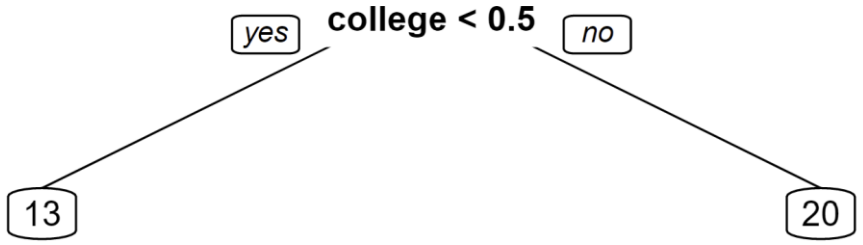


Figure 2.2: Depth 1 tree in the wage example

Growing Regression Trees

- Regression trees are typically constructed in a “greedy” manner
- We then go to each node and locally we find the regressor and threshold on which to split that leads to largest improvement in MSE

- We repeat until some stopping criterion (e.g. total number of nodes, minimum number of leaf samples, maximum depth)

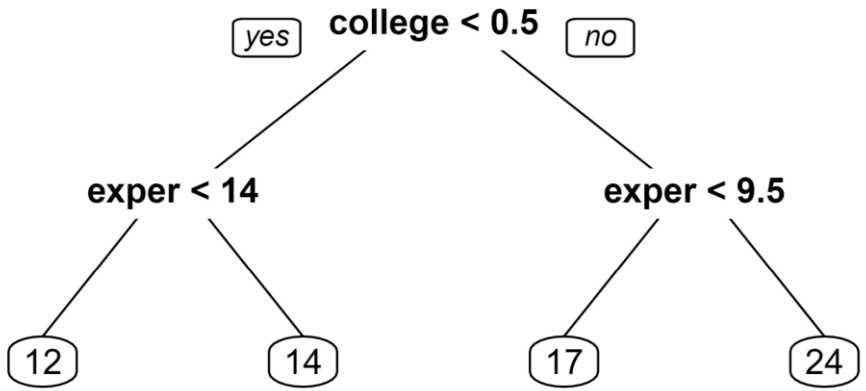


Figure 2.3: Depth 2 tree in the wage example

Regression Trees: Bias-Variance Tradeoffs

- The deeper the tree, the better it can approximate the CEF (small bias)
- The deeper the tree, the noisier the estimate is, as we have very few samples in each leaf (higher variance)

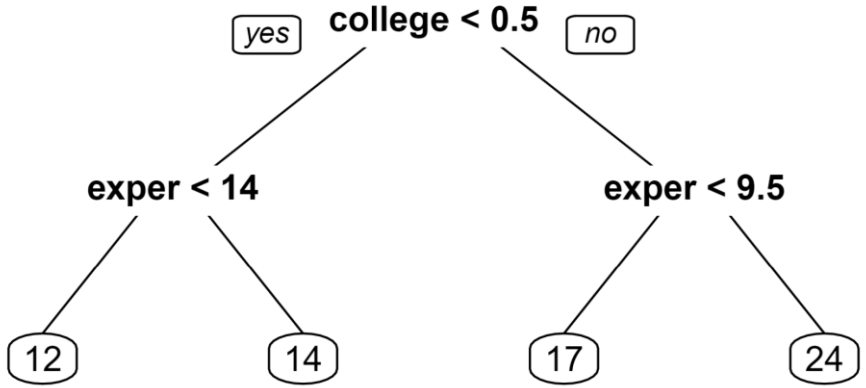


Figure 2.3: Depth 2 tree in the wage example

Let's check it out!

Some Theory

- Suppose that all variables are binary (or categorical)
- Without any assumption, at best we can achieve error $\sqrt{\frac{2^p}{n}}$
- Suppose that the CEF depends only on $s \ll p$ "relevant" variables

$$g(Z) = f(Z_R), \qquad |R| = s$$

- s is the "effective dimension"

Theorem [Syrgkanis-Zampetakis'20]. Under several regularity conditions, greedily grown regression trees with max depth at least s and at most a multiple of s achieve error $\approx \sqrt{\frac{2^s \log(p) \log(n)}{n}}$

Problems with Trees

- They tend to find very discontinuous approximations to the true CEF
- Real world CEFs are most times smooth functions
- We need to smoothen the output of a regression tree
- Basic idea: average over multiple trees, each built by injecting some randomness

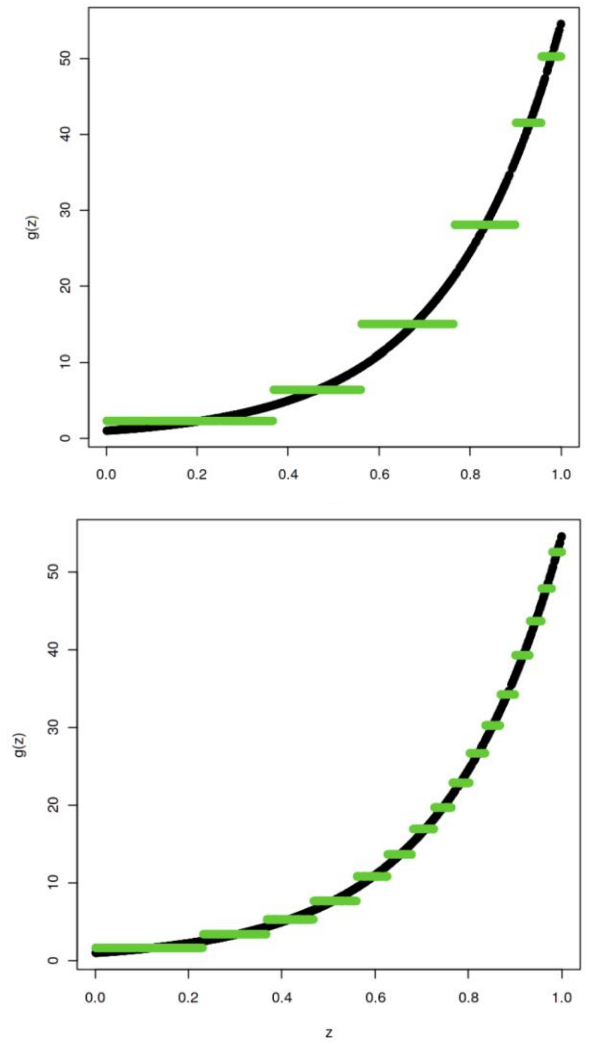
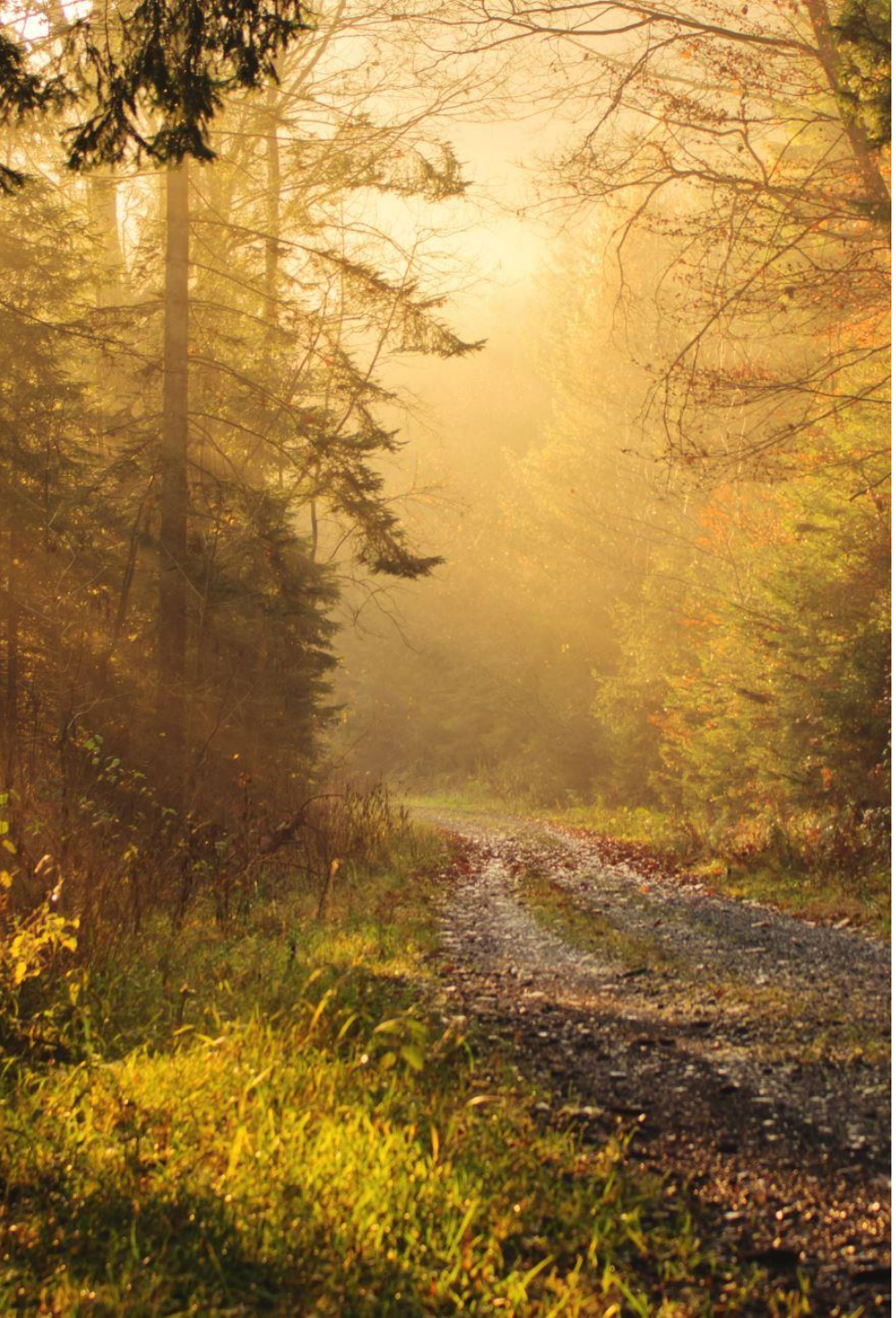


Figure 2.7: Approximation of $g(Z) = \exp(4Z)$ by a deep Regression Tree in the noiseless case.



Regression Forests

- Construct many trees $g_1, \ldots, g_b, \ldots, g_B$ (forest)
- Each tree $\hat{g}_b$ built using only a random subset S_b of the data
- Typical practice, *bootstrapping*: draw $\approx n$ observations uniformly at random from the data **with replacement**
- Typical theory, *subsampling*: draw $s \ll n$ observations uniformly at random from the data **without replacement**
- Typically, extra randomness is injected (e.g. random subset of variables is drawn as candidate splits at each node)

- Final prediction is average of predictions of each of the trees

Bagging

- Draw multiple random subsets of the data with replacement $S_1, \ldots, S_B$
- Train a base model $\hat{g}_b$ using only data from S_b
- Return the "ensemble" or "average prediction rule"

$$\hat{g}(z) = \frac{1}{B}\sum_{b=1}^{B} \hat{g}_b(z)$$

- Can be performed with any base model
- Frequently used base model are decision trees
- Leads to smoother functions and reduces variance

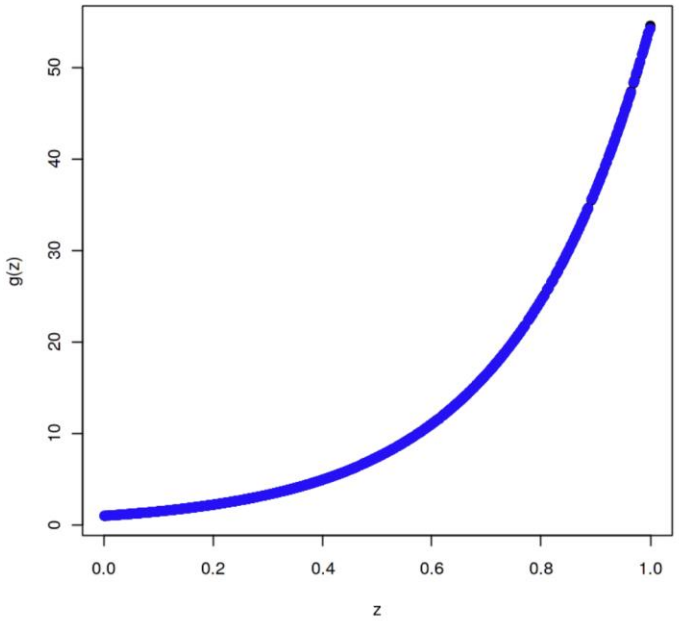


Figure 2.8: Approximation of $g(Z) = \exp(4Z)$ by a Random Forest in the noiseless case.

Let's check it out!

Other knobs of regularization

- *Subsampling.* typically leads to even better variance reduction, especially for very deep trees

- *Honesty.* construct splits based on half the data and estimate the values at the leaf nodes using the other half; can help a lot in preventing over-fitting

Let’s check it out!

Some Theory

- Suppose that all variables are binary (or categorical)
- Without any assumption, at best we can achieve error $\sqrt{\frac{2^p}{n}}$
- Suppose that the CEF depends only on $s \ll p$ "relevant" variables

$$g(Z) = f(Z_R), \qquad |R| = s$$

Theorem [Syrgkanis-Zampetakis'20]. Under several regularity conditions, greedily grown regression forests with *deep and honest trees built on sub-samples* of size $\approx 2^s \log(p)$ achieve error $\approx \sqrt{\frac{2^s \log(p)\, \text{polylog}(n)}{n}}$

Problems with Bagging

- The different base models are trained without “knowledge” of what the other models learned
- Maybe we want to build base models incrementally
- Every time we try to capture some variation/pattern in the outcome that has not been captured so far

- This is the idea of Gradient Boosting!

Gradient Boosting

Initialize the "residual outcome variation" aka "residuals"

$$R_i = Y_i$$

At each step b:

- fit a base model $\hat{g}_b$ predicting the residuals R_i from Z_i
- Update residuals to remove part of newly explained variation\pattern:

$$R_i \leftarrow R_i - \lambda \hat{g}_b(Z_i)$$

Finally return "boosted ensemble" or "boosted prediction rule"

$$\hat{g}(z) \coloneqq \sum_{b=1}^{B} \lambda \hat{g}_b(z)$$

Gradient Boosted Forests

- Gradient boosting can be applied with any base model
- Frequently used base model are ***simple*** decision trees
- Takes simple functions and “boosts” their approximation capabilities
- ***Unlike bagging:*** typically leads to smaller bias but higher variance
- Best way to control variance increase is to perform “early stopping”
- At each step measure performance of current ensemble on a validation set
- Stop when you see that validation set performance increases

Let’s check it out!

Gradient Boosted Forests

- Theory less well developed in high-dimensions
- Some very nice theoretical results proving consistency and adaptivity to notions of statistical complexity; *early stopping is crucial*

Some pointers

- Beygelzimer, Alina, et al. "Online gradient boosting." Advances in neural information processing systems 28 (2015).
- Zhang, Tong, and Bin Yu. "Boosting with early stopping: Convergence and consistency." (2005): 1538-1579.
- Wei, Yuting, Fanny Yang, and Martin J. Wainwright. "Early stopping for kernel boosting algorithms: A general analysis with localized complexities." Advances in Neural Information Processing Systems 30 (2017).

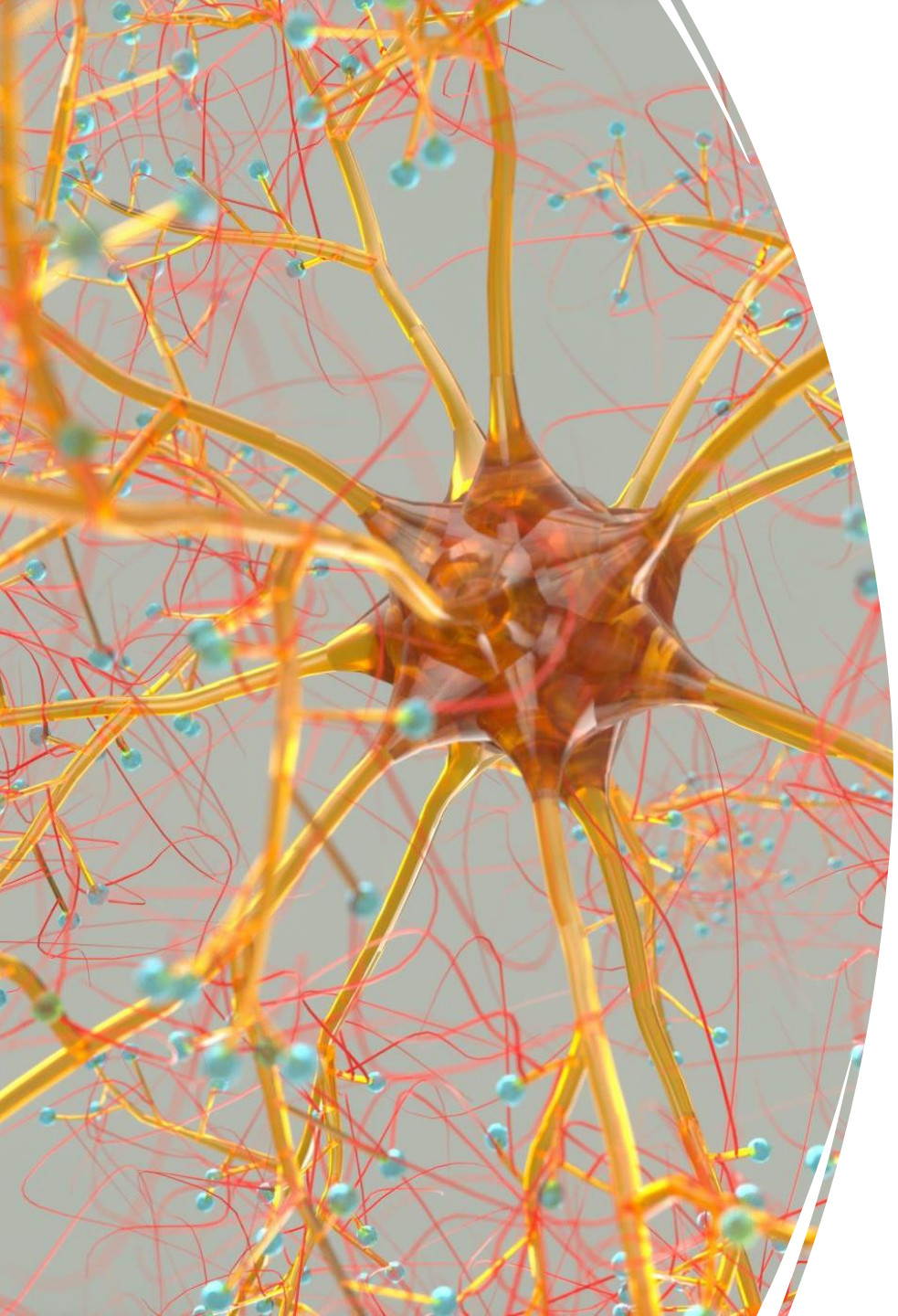
Problems with Forests and Trees

- They are based on simple adaptively learned engineered features (indicators of rectangles)
- Why not learn more complex features from the data
- Why not learn a linear model

$$g(z) = \beta' \phi(z; \alpha)$$

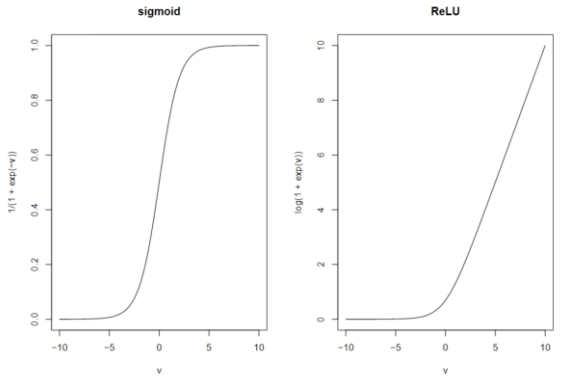
- Where the features themselves are parameterized by some parameter α and learned also from the data?

- This leads to neural networks



(Shallow) Neural Networks

- We approximate the CEF with data-driven engineered features
$$g(z) \coloneqq \beta' \phi(z; a)$$
- Typical choice of ϕ is:
$$\phi(z; a) = \sigma(a'z)$$
- With σ some non-linear function



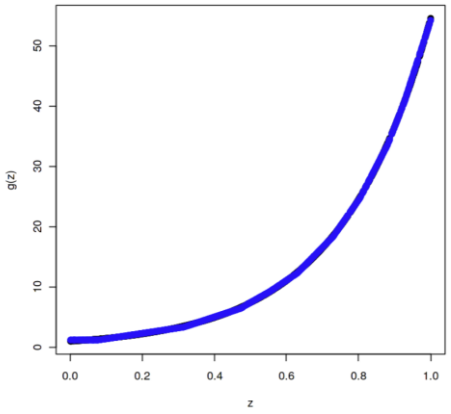


Figure 2.12: Approximation of $g(Z) = \exp(4Z)$ by a Neural Network

(Shallow) Neural Network Objective

- Parameters chosen by minimizing penalized empirical square loss

$$\min_{\alpha,\beta} E_n\left[\left(Y - \beta'\phi(Z;\alpha)\right)^2\right] + \lambda\,\mathrm{pen}(a,\beta)$$

- Penalty is either ℓ_1 norm (sparsity inducing) or ℓ_2 norm (inducing small weights); λ is referred as *weight decay* in the case of ℓ_2
- Loss is typically minimized via *Stochastic Gradient Descent* (SGD)

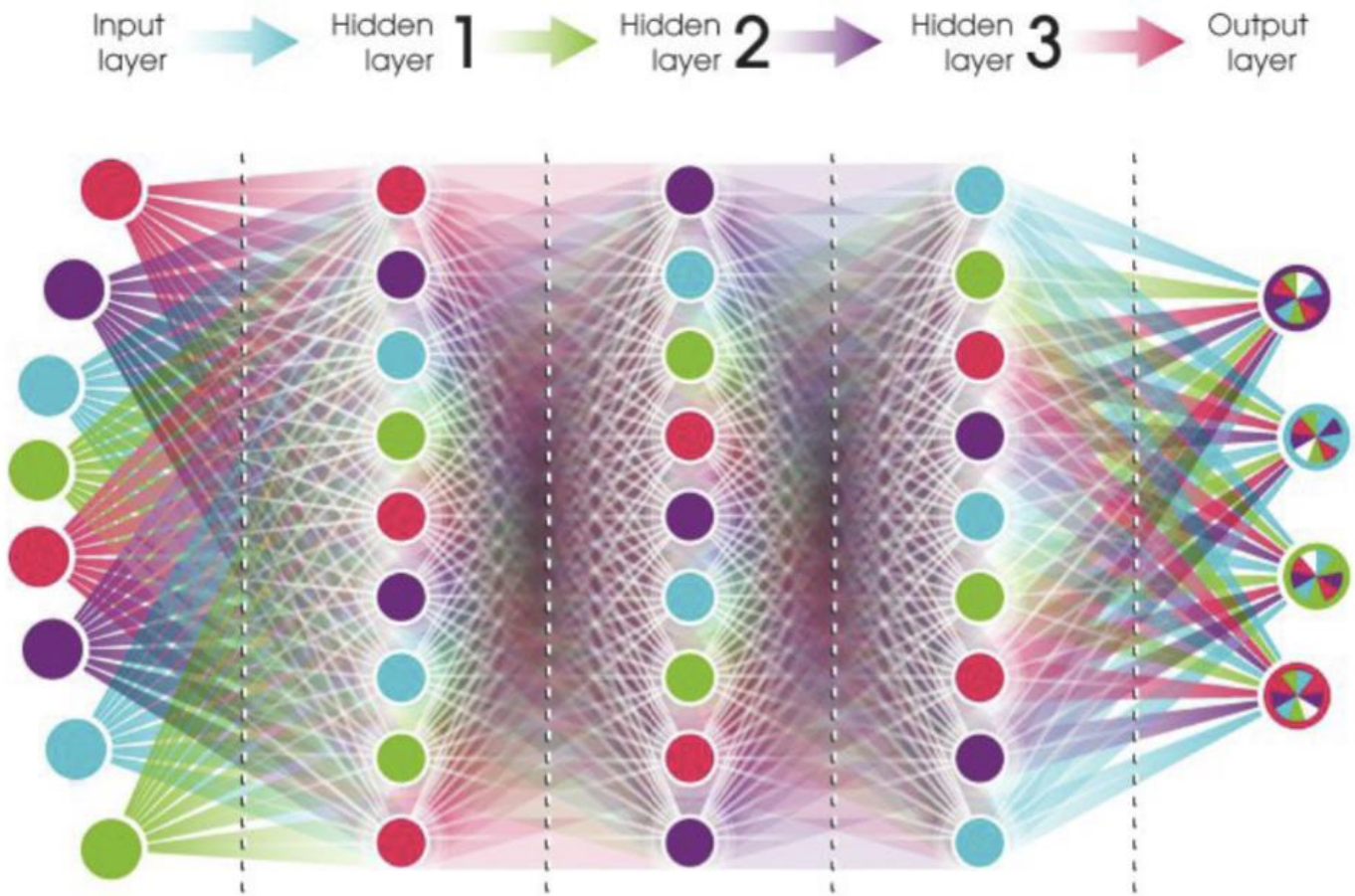
$$(\alpha,\beta) \leftarrow (\alpha,\beta) - \eta\partial_{\alpha,\beta}\mathrm{Loss}(\mathrm{B};\alpha,\beta)$$

- $\mathrm{Loss}(\mathrm{B};\alpha,\beta)$ is empirical loss calculated on a sub-sample B (*batch*)

$$\frac{1}{|B|}\sum_{i\in B}\left(Y_i - \beta'\phi(Z_i;\alpha)\right)^2 + \lambda\,\mathrm{pen}(a,\beta)$$

- Every pass over all the data is referred to as an *epoch*

DEEP NEURAL NETWORK



neuralnetworksanddeeplearning.com - Michael Nielsen, Yoshua Bengio, Ian Goodfellow, and Aaron Courville, 2016.

Forms of Regularization

- *Stochasticity* and iterative nature of SGD is by itself a regularization method (implicit regularization)
- *Penalties* are explicit form of regularization
- *Drop-out.* At each training step, shutdown some of the neurons. Implicitly regularizes by having multiple neurons learn important concepts, acting as substitutes
- *Early stopping.* Measure out of sample performance after a few iterations of SGD and stop if it stops improving

Let’s check it out!

Some Theory

Structured Sparsity and Smoothness. Assume g is a composition

$$g = f_M \circ \cdots \circ f_0$$

Where i-th function $f_i : \mathbb{R}^{p_i} \to \mathbb{R}^{p_{i+1}}$ has its p_{i+1} components β_i-smooth (*smoothness*) and depends only $t_i \ll p_i$ input variables (*sparsity*)

Effective dimension is $s := \max_i n^{-\frac{\beta_i}{2\beta_i + p_i}}$

Theorem[Schmidt-Hieber'20]. If depth$\sim \log(n)$ and width $\geq s\log(n)$, and several other regularity conditions, then error of an appropriately trained neural network is at most $\approx \sqrt{\frac{s}{n}}\,\mathrm{polylog}(n)$

A Reminder: Fancy isn't always better

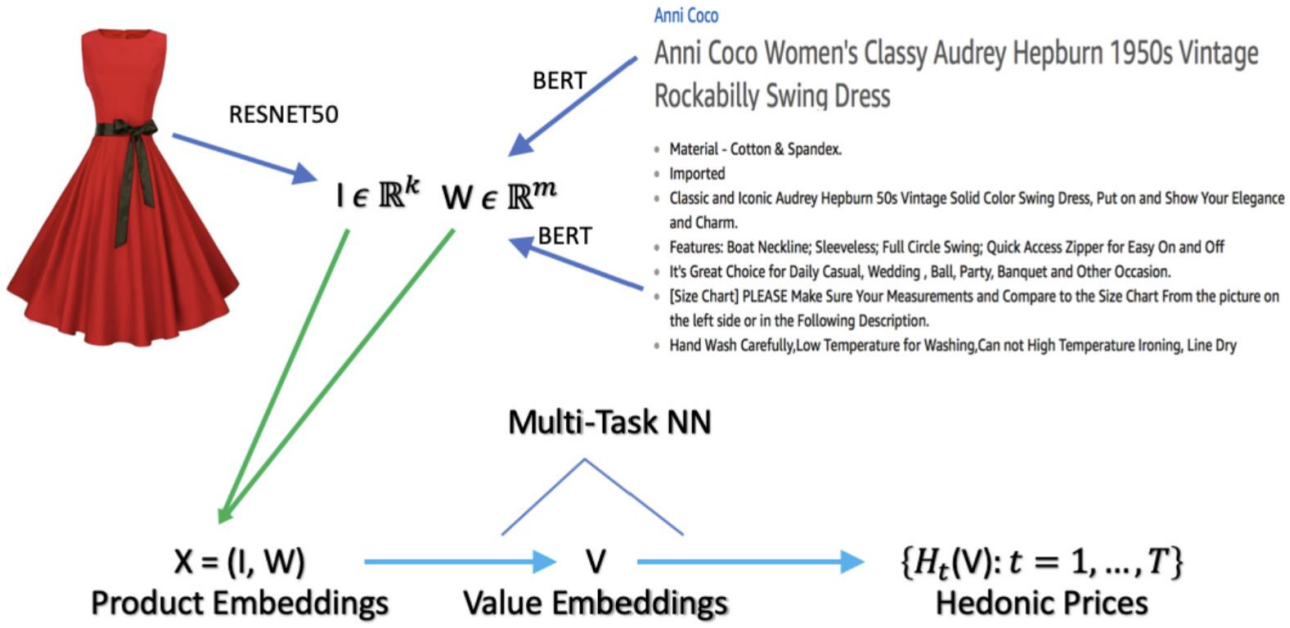
Predictive performance for predicting wages

	MSE	S.E.	R^2
Least Squares (basic)	0.229	0.016	0.282
Least Squares (flexible)	0.243	0.016	0.238
Lasso	0.234	0.015	0.267
Post-Lasso	0.233	0.015	0.271
Lasso (flexible)	0.235	0.015	0.265
Post-Lasso (flexible)	0.236	0.016	0.261
Cross-Validated Lasso	0.229	0.015	0.282
Cross-Validated Ridge	0.234	0.015	0.267
Cross-Validated Elastic Net	0.230	0.015	0.280
Cross-Validated Lasso (flexible)	0.232	0.015	0.275
Cross-Validated Ridge (flexible)	0.233	0.015	0.271
Cross-Validated Elastic Net (flexible)	0.231	0.015	0.276
Random Forest	0.233	0.015	0.270
Boosted Trees	0.230	0.015	0.279
Pruned Tree	0.248	0.016	0.224
Neural Net	0.276	0.012	0.148

But many times it is crucial

Predicting prices from product characteristics at Amazon

Bajari et al. 2021, *Hedonic prices and quality adjusted price indices powered by AI*.



Which method should I use?

Use all and combine: Stacking

- If you have many models $\hat{g}_1, \ldots, \hat{g}_K$ we can combine based on out-of-sample performance

$$\text{Best} - \text{Loss} = \max_k E\left[\left(Y - \hat{g}_k(Z)\right)^2\right]$$

$$= \max_{w \geq 0 : \sum_k w_k = 1} \sum_k w_k E\left[\left(Y - \hat{g}_k(Z)\right)^2\right]$$

$$\geq \max_{w \geq 0 : \sum_k w_k = 1} E\left[\left(Y - \sum_k w_k \, \hat{g}_k(Z)\right)^2\right] = \text{Loss of Best Ensemble}$$

Stacking

- Train an OLS on the out-of-sample data predicting Y with features $g_1(Z), \ldots, g_K(Z)$ to learn weights w; return ensemble prediction

$$\hat{g}(Z) \coloneqq \sum_{k=1}^{K} w_k \hat{g}_k(Z)$$

- If models are too many, we can train Lasso on out-of-sample to learn weights, to avoid overfitting!

Let's check it out!

How do I choose all these hyperparameters?

Use Auto-ML frameworks!

- Automatic and clever search over the hyperparameter space
- Very few lines of code
- Typically much better performance than handpicking yourself
- Unless a lot of domain knowledge of what types of functions are better approximators

- Many user-friendly tools: H2O-AutoML, Auto-Gluon, Azure-AutoML, FLAML, Auto-Sklearn, HyperOpt-Sklearn