# Big Data Asset Pricing

## Lecture 5: Machine Learning in Asset Pricing

Lasse Heje Pedersen
*AQR, Copenhagen Business School, CEPR*

https://www.lhpedersen.com/big-data-asset-pricing

The views expressed are those of the author
and not necessarily those of AQR

# Overview of the Course: Big Data Asset Pricing

## Lectures

- Quickly getting to the research frontier
  1. A primer on asset pricing
  2. A primer on empirical asset pricing
  3. Working with big asset pricing data (videos)
- Twenty-first-century topics
  4. The factor zoo and replication
  5. **Machine learning in asset pricing**
  6. Asset pricing with frictions

## Exercises

1. Beta-dollar neutral portfolios
2. Construct value factors
3. Factor replication analysis
4. High-dimensional return prediction
5. Research proposal

# Overview of this Lecture

- Overview of machine learning
  - Supervised: regression vs. classification
  - Unsupervised
- How to apply ML
  - Bias-variance trade-off
  - Hyper-parameters
  - Train, validation, test
- Specific ML models in more detail
  - Penalized regressions
  - Regression trees
  - Neural networks
  - Feature importance
- Applications to asset pricing
  - Predicting stock returns
  - What factors generate the pricing kernel?

# Overview of Machine Learning

# Supervised vs. Unsupervised Learning

- Supervised learning: start with inputs and outputs
  - Terminology:

| | Terminology | | Notation | |
|---|---|---|---|---|
| | Statistics | ML | Stat/ML | Finance |
| Inputs | Independent variables | Features | $x^i$ | $s_t^i$ |
| Outputs | Dependent variable | Response | $y^i$ | $r_{t+1}^i$ |

  - Two types of supervised learning: classification vs. regression

| | Types of outputs | # of outputs | Example |
|---|---|---|---|
| Classification | Qualitative | Finite | Default/no default |
| Regression | Quantitative | Any value | Predict return |

- Unsupervised: start with just inputs (no outputs)
  - Example: clustering of factors into groups

# Classic ML Problem: Supervised Learning via Regression

- Start with data on inputs, $x^i \in \mathbb{R}^K$, and outputs, $y^i \in \mathbb{R}^1$
- Training set of observations, $i = 1, ..., T$
- Want to learn $f$ such that

$$y^i = f(x^i) + \varepsilon^i$$

  - Think of $f$ as $f(x^i) = E(y^i|x^i)$, projection, or just a way to guess the output
- Ways to learn about $f$:

| Assumption | Statistics | Finance | ML |
|---|---|---|---|
| $f(x^i) = \text{ave}(y^j)_{x^j \text{ near } x^i}$ | Kernel regression | Portfolio sort | Regression trees, random forest, boosting |
| $f(x^i)$ linear | OLS, GLS | Fama-MacBeth, regression-based mkt timing, signal-weighted portfolio | Penalized regression, e.g., LASSO, ridge |
| $f(x^i)$ non-linear | Non-linear least squares, GMM, MLE | | Neural networks, deep learning |

# How to Apply ML

# Leading Example: Ridge Regression

- Want to learn $f$ such that $y^i = f(x^i) + \varepsilon^i$
  - Regression: $y^i = x^i\beta + \varepsilon^i$
  - Vector form: $y = x\beta + \varepsilon$
- OLS: $\hat{\beta} = (x'x)^{-1}x'y$
  - Properties: $\hat{\beta} = (x'x)^{-1}x'y = \beta + (\frac{1}{T}x'x)^{-1}\frac{1}{T}x'\varepsilon \to \beta$ for $T \to \infty$
  - What does $T \to \infty$ mean in practice? That $T$ is large relative to $K$
    - But $K$ is large when we do ML
    - Estimating many regression coefficients creates a lot of noise
  - Gauss-Markov theorem: OLS has smallest mean squared error of all linear estimators with no bias
    - But we can do better *with* bias
    - By reducing estimation noise – more on this later
- Ridge regression
  - Objective: minimize $\frac{1}{T}(y - x\beta)'(y - x\beta) + \lambda\beta'\beta$
  - Solution: $\hat{\beta}^{\text{ridge}} = (\frac{1}{T}x'x + \lambda I)^{-1}\frac{1}{T}x'y$
  - But, how do we choose (or "tune") $\lambda$?

# Parameters vs. Hyper-Parameters (or Tuning Parameters)

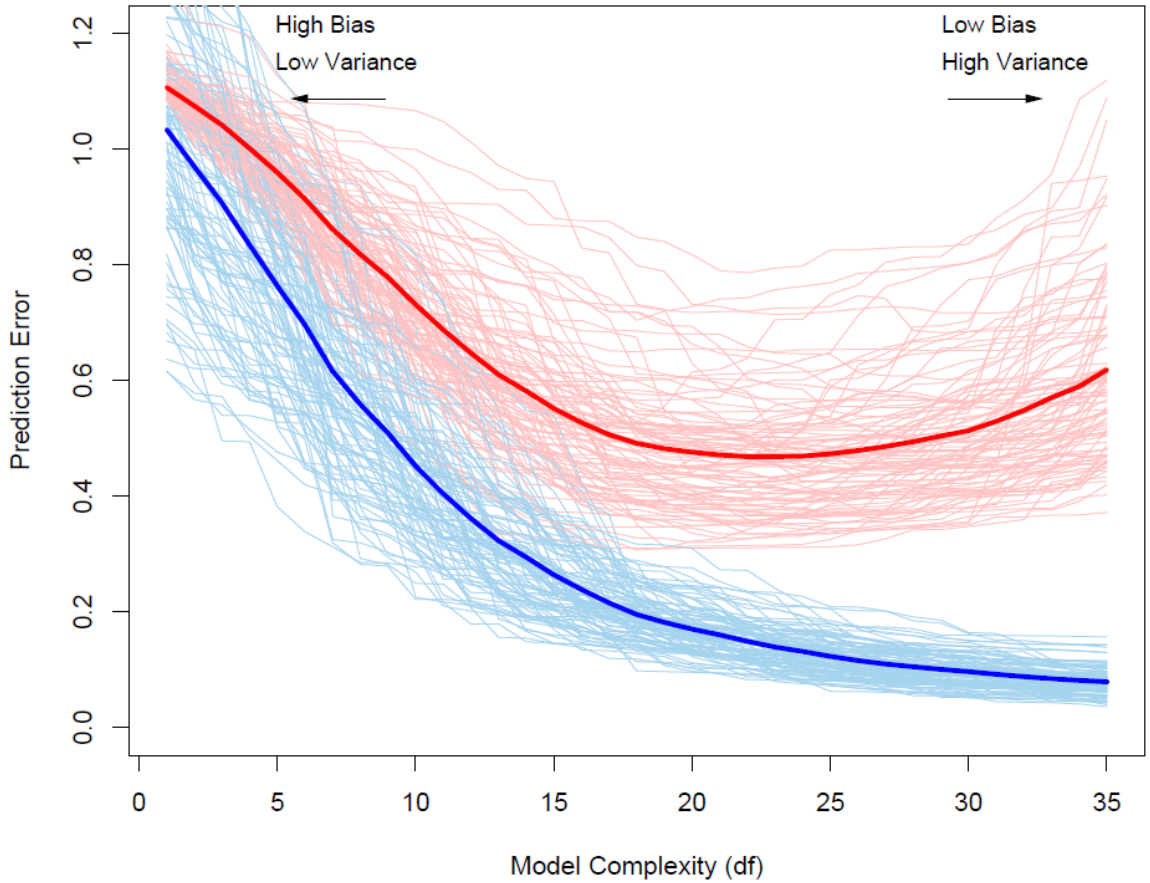
- Want to learn $f$ such that

$$y^i = f(x^i) + \varepsilon^i \overset{\text{example}}{=} x^i\beta + \varepsilon^i$$

- Leading example, ridge regression, $\hat{\beta}^{\text{ridge}} = (\frac{1}{T}x'x + \lambda I)^{-1}\frac{1}{T}x'y$
- Parameters and hyper-parameters:
  - Parameters tell us what $f$ is: $\hat{f}(x^i) = f(x^i;\hat{\beta}) \overset{\text{example}}{=} x^i\hat{\beta}^{\text{ridge}}$
  - Hyper-parameters tell us how to find the parameters, e.g., $\lambda$
- Terminology: hyper-parameters = tuning parameters
- Parameters and hyper-parameters: other examples

| Method | Parameters | Hyper-parameters |
|---|---|---|
| Penalized regression | Regression coefficients, $\beta$ | Penalty $\lambda$ for large $\beta$ |
| Regression tree | Tree splits | Tree depth, etc. |
| Neural network | Coefficients | Layers, etc. |

# Bias-Variance Trade-Off




Source: Hastie et al. (2009). The light blue curves: training error. Light red: conditional test error.

- Example: ridge regression
  - Smaller $\lambda$ = greater model complexity = more degrees of freedom
  - More complexity always leads to better fit in the training sample
  - More complexity leads to lower bias, but higher variance
  - Best performance in the test period for intermediate $\lambda$

# Bias-Variance Trade-Off, continued

- Prediction error:
  - Prediction: $\hat{y}^i = \hat{f}(x^i)$
  - Outcome, out-of-sample: $y^i = f(x^i) + \varepsilon^i$
  - Prediction error: $\hat{f}(x^i) - y^i$
- Expected "loss," $L$ (squared prediction error, or absolute prediction error):

$$\begin{aligned}
\text{E}[L] = \text{E}[\text{prediction error}^2] =& \text{E}[(\hat{f}(x^i) - y^i)^2] \\
=& \text{E}[(\hat{f}(x^i) - f(x^i) - \varepsilon^i)^2] \\
=& \sigma_\varepsilon^2 + \text{E}[(\hat{f}(x^i) - f(x^i))^2] \\
=& \sigma_\varepsilon^2 + (\text{E}[\hat{f}(x^i)] - f(x^i))^2 + \text{E}[(\hat{f}(x^i) - \text{E}[\hat{f}(x^i)])^2] \\
=& \text{noise} + \text{bias}^2 + \text{variance}
\end{aligned}$$

  - **Noise**: from $\varepsilon$, unavoidable, the same regardless of prediction method
  - **Bias**: Tendency of the prediction to miss its target even with lots of data
  - **Variance**: $\text{Var}(\hat{f}(x^i))$ comes from noise in parameter estimation
- Bias-variance trade-off:
  - Lower complexity, e.g., shrinking $\hat{\beta}$ more toward zero
    - Increases bias
    - Lowers variance
- So, an intermediate $\lambda$ is optimal, but how to choose it?

# Training, Validation, and Testing

- Old school finance and statistics: 1 sample (everything in-sample)
- Finance and statistics with model assessment: 2 sub-samples
  1. **In-sample**: estimate parameters
  2. **Out-of-sample**: assess performance
- ML: 3 sub-samples
  1. **Train**: estimate parameters for each hyper-parameter
  2. **Validation**: pick the best-performing hyper-parameter
  3. **Test**: assess performance
- ML: example of ridge regression
  1. **Train**: estimate $\hat{\beta}^{\text{ridge}}(\lambda) = (x'x + \lambda I)^{-1}x'y$ for each $\lambda$
  2. **Validation**: pick $\hat{\lambda}$ with small loss, $L[y^i - x^i\hat{\beta}^{\text{ridge}}(\hat{\lambda})]$
  3. **Test**: assess performance of $\hat{\beta}^{\text{ridge}}(\hat{\lambda})$

# Cross-Validation

- ***k*-fold cross-validation**
  - Data scarce – want to use full train+validation sample for both
  - Split train+validation sample into $k$ folds (subsets), but note
    - $k$ is not the same as the number of features $K$
    - In finance, ensure that each whole cross-section is in the same fold, but standard cross-validation functions distribute data randomly
- **Example: 5-fold validation**

| | Data | | | | |
|---|---|---|---|---|---|
| Split 1 | Validate | Train | Train | Train | Train |
| Split 2 | Train | Validate | Train | Train | Train |
| Split 3 | Train | Train | Validate | Train | Train |
| Split 4 | Train | Train | Train | Validate | Train |
| Split 5 | Train | Train | Train | Train | Validate |

- **Method:**
  1. **Train $k$ times:** For each split $j$ and each hyper-parameter
     - estimate parameters using training data in all folds, except $j$
     - compute prediction error in $j$'th fold, i.e., $j$'th validation period
  2. **Cross-validate**: Choose hyper-parameters
     - that minimize prediction errors across all validation periods
  3. **Retrain**: estimate parameters using full train+validation data
  4. **Test**: assess performance (as always)

# Another Validation Method that Reuses the Data

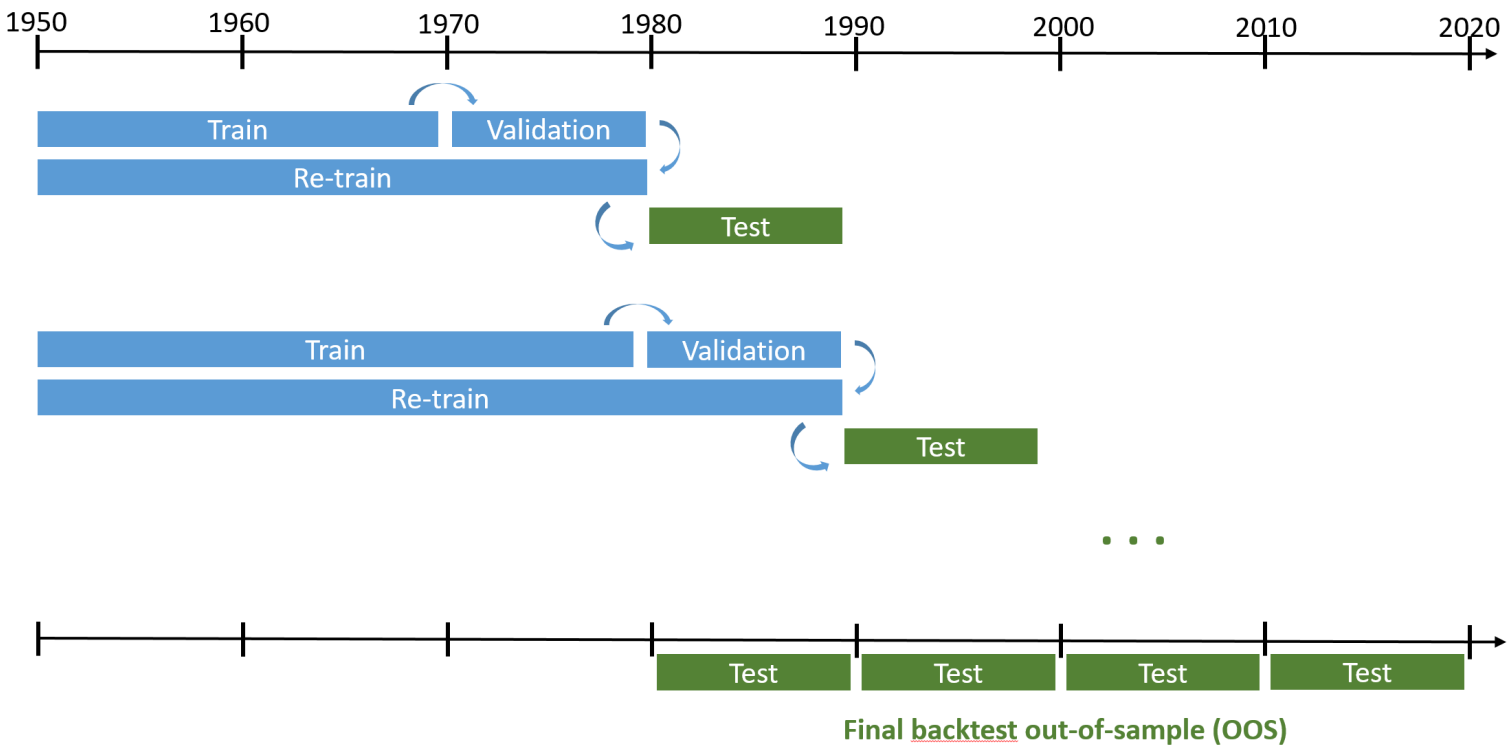
- When the data has a time-series dimension
  - finance papers usually keep the temporal order such that the validation set comes after the training set
  - e.g., in asset pricing, people often use rolling estimation
  - not necessarily wrong to reverse order
- First do the standard ML split into 3 sub-samples
  1. Train
  2. Validation
  3. Test
- Then proceed in these steps
  1. Train: estimate $\hat{\beta}$ for each $\lambda$ in the training sample
  2. Validation: pick the best-performing $\hat{\lambda}$
  3. Possibly re-train:
     - Estimate $\hat{\beta}$ for $\hat{\lambda}$ in joint sample: training+validation
     - NB: re-training requires that $\hat{\lambda}$ "immune" to sample length
       E.g., the $T$ matters in $\min_\beta \frac{1}{T}(y - x\beta)'(y - x\beta) + \hat{\lambda}\beta'\beta$
  4. Test: assess performance

# Expanding Training Set

In finance, people often use a rolling or expanding training set

- Expanding: always start from the beginning of the sample
- Rolling: Use a fixed number of years (discarding the oldest data)

**Illustration of expanding training set:**



# Specific ML Models: Penalized Regressions

For details, see excellent and free books Hastie et al. (2009) and Efron and Hastie (2021)

# Penalized Regressions: Lasso

- Want to learn $f$ such that $y^i = f(x^i) + \varepsilon^i$
  - Regression: $y^i = x^i\beta + \varepsilon^i$
  - Vector form: $y = x\beta + \varepsilon$
- **Ridge regression**: we already saw this
- **Lasso regression**:
  - Objective:

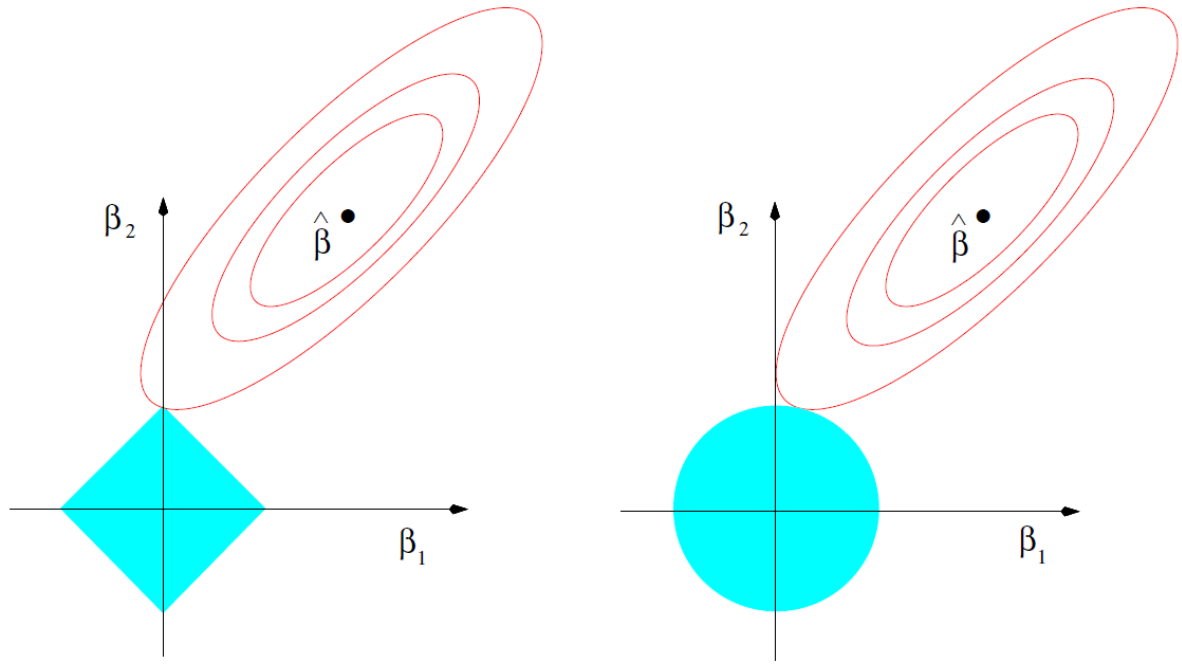
$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left( \frac{1}{T}(y - x\beta)'(y - x\beta) + \lambda \sum_j |\beta_j| \right)$$

  - Alternative objective:

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \frac{1}{T}(y - x\beta)'(y - x\beta)$$
$$\text{subject to } \sum_j |\beta_j| \leq \bar{\lambda}$$

- **Cf. best subset**: use OLS for subset of $x$ variables (less used)

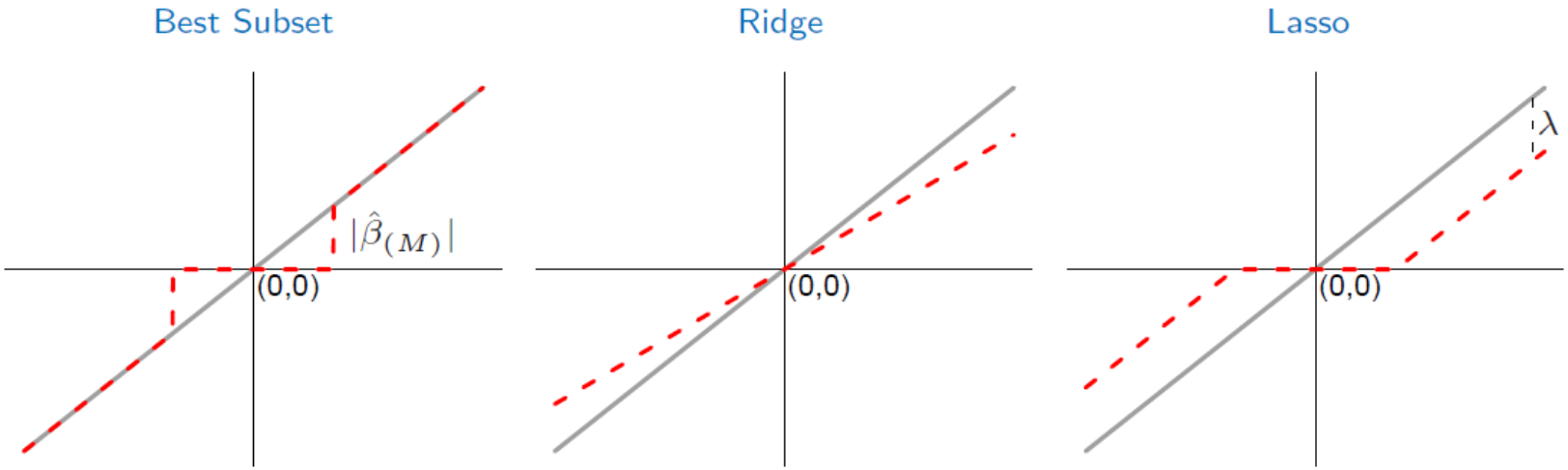
# Penalized Regressions: Comparison




Source: Hastie et al. (2009). Lasso regression (left) and ridge regression (right). Red lines show contours of the prediction errors, $(y - x\beta)'(y - x\beta)$. The solid blue areas show, respectively, subject to $\sum_j |\beta_j| \leq \bar{\lambda}$ and subject to $\sum_j \beta_j^2 \leq \bar{\lambda}$

- Differences
  - Ridge shrinks all parameters, but generically all parameters are non-zero
  - Lasso sets some parameters to zero, here $\beta_1 = 0$, and shrinks the rest

# Penalized Regressions: Comparison, continued




Source: Hastie et al. (2009). The $x$-axis has the OLS estimate, $\hat{\beta}$, and the $y$-axis has the respective penalized regression estimates.

- Differences
  - Best subset: once a variable is included, there is no shrinkage in its beta
  - Ridge: proportional shrinkage, with one parameter:

$$\hat{\beta}^{\text{ridge}} = (\frac{1}{T}x'x + \lambda I)^{-1}\frac{1}{T}x'y = \frac{\hat{\beta}^{OLS}}{\tilde{\lambda}} \text{ , where } \tilde{\lambda} = 1 + \lambda/(\frac{1}{T}x'x)$$

  - Lasso: once beta is non-zero, it is shrunk by a constant, e.g. if $\beta > 0$

$$\hat{\beta}^{\text{lasso}} = (\frac{1}{T}x'x)^{-1}\left(\frac{1}{T}x'y - \frac{1}{2}\lambda\right) = \hat{\beta}^{OLS} - \tilde{\lambda} \text{ , where } \tilde{\lambda} = \frac{1}{\frac{2}{T}x'x}\lambda$$

# Penalized Regressions: Elastic-Net

- Want to learn $f$ such that $y^i = f(x^i) + \varepsilon^i$
  - Regression: $y^i = x^i\beta + \varepsilon^i$
  - Vector form: $y = x\beta + \varepsilon$

- **Elastic-net regression**:

$$\hat{\beta}^{\text{elastic-net}} = \arg\min_{\beta} \left( \frac{1}{T}(y - x\beta)'(y - x\beta) + \lambda \sum_j \left( a\beta_j^2 + (1-a)|\beta_j| \right) \right)$$

- Two tuning parameters:
  - $\lambda \geq 0$ controls amount of regularization
  - $a \in [0, 1]$ controls elements of ridge vs. lasso
- Some betas are set to zero, all are shrunk

# Specific ML Models:
# Regressions Trees and Random Forests

# Regression Trees

What is a regression tree?

- Partition the space of features into $J$ regions, $R_1, ..., R_J$
- Estimate response function as

$$\hat{f}_{\text{tree}}(x) = \sum_j 1_{\{x \in R_j\}} \text{ave}(y_i | x_i \in R_j)$$

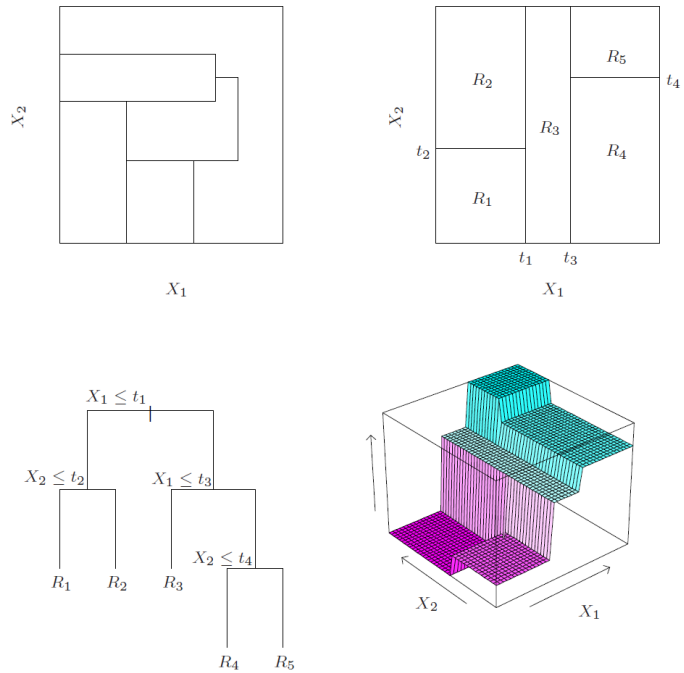
"Greedy" algorithm to make a tree of maximum depth $D$:

- Start with a tree of depth $d = 0$ (all data in one region)
- For $d = 1, ..., D$, make tree of depth $d$ from prior tree of depth $d - 1$ by splitting each region as follows:
  - If a region is smaller than a minimum size, leave as is
  - Otherwise, consider splitting the region into two:
    1. For each feature $k = 1, ..., K$, find the split point $s(k)$, that leads to the smallest prediction error
    2. Pick the feature $k^*$ such that the split $[k^*, s(k^*)]$ leads to smallest prediction error
    3. Split the region into two daughter regions based on $[k^*, s(k^*)]$ unless one of daughter regions has too few elements or the improvement in prediction error is too small (alternatively, prune afterward)

Hyper-paramaters:

- Tree depth, minimum observations per region, minimum loss improvement, etc.

# Regression Trees: Generic Example



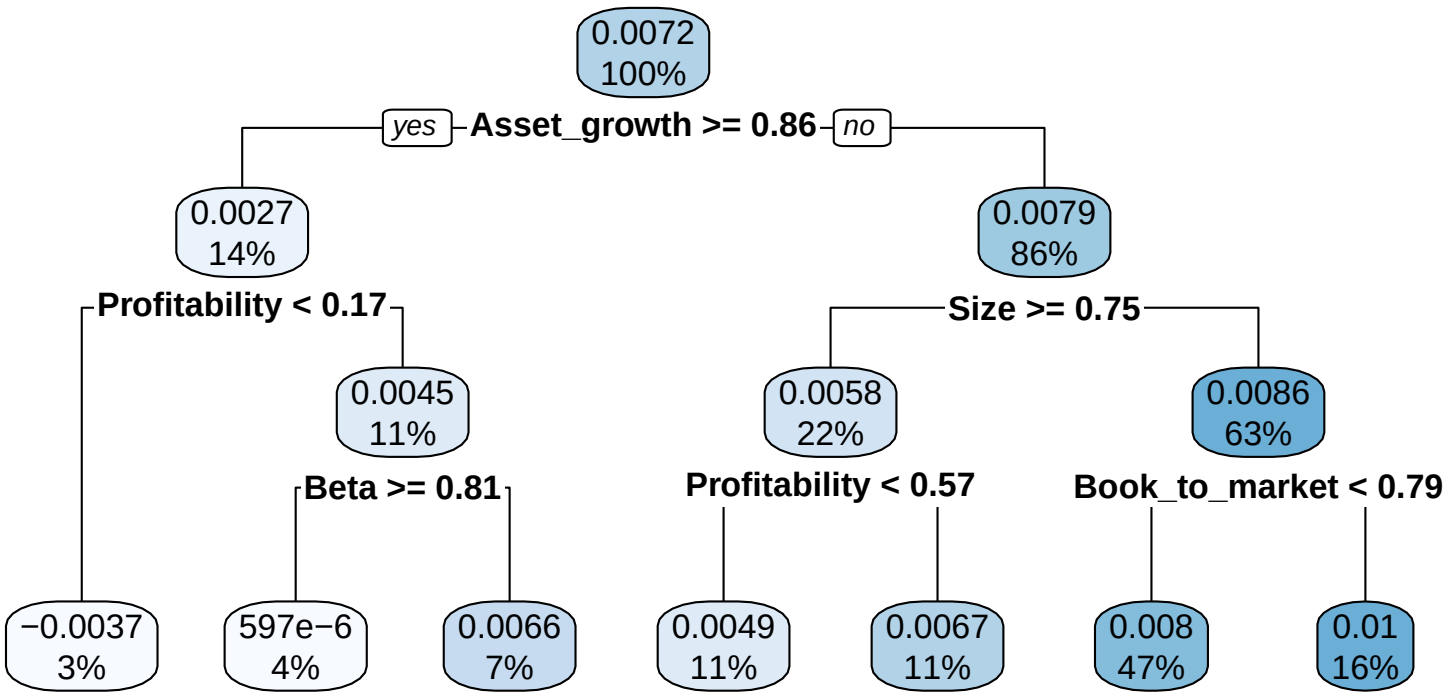

Source: Hastie et al. (2009). Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.

- Greedy algorithm
  - solve problem in stages, making the locally optimal choice at each stage
  - does not necessarily produce an optimal solution, but fast
- Here: always split each region (or branch) into two (or leave as is)

# Regression Trees: Empirical Example from Asset Pricing

Use regression tree to predict excess returns

- Sample: Non-microcap stocks in the US
- Features: Beta, book-to-market, market equity, profitability, and asset growth, all standardized to cross-sectional percentiles
- Hyper-parameters: Max depth=3 and no split if leaf has <10% of obs
- Each circle contains that node's
  - average excess return
  - percent of observations



# Regression Trees: Advantages and Disadvantages

- Advantages
  - Relatively simple and easy to interpret
  - Insensitive to monotone transformations of predictors
  - Can capture interactions of predictors (up to depth minus 1)
  - Can handle missing feature values
- Disadvantages
  - Not smooth
  - Possible instability and sensitivity to a few observations
  - I.e., high variance

# Random Forests and Bagging

- Bagging=Bootstrap aggregating
  - ML method to improve the stability and accuracy
  - Model averaging of models fitted to bootstrap samples
- Random forest = bagging applied to regression trees
- How to make a random forest:
  - Draw $b = 1, ..., B$ different bootstrap samples of the data
  - Fit a separate regression tree to each bootstrap $b$
    - To make these trees less correlated, each split is based on a random subset of features
  - Average the forecasts coming from each tree

$$\hat{f}_{\text{random forest}}(x) = \frac{1}{B} \sum_{b} \hat{f}_{\text{tree},b}(x)$$

# Gradient Tree Boosting

- Gradient boosting with quadratic loss, $L(y^i - \hat{y}^i) = \frac{1}{2}(y^i - \hat{y}^i)^2$
- How to reduce the loss with $\hat{y}^i = \hat{f}(x^i)$? Answer:

$$-\frac{\partial L}{\partial \hat{y}^i} = y^i - \hat{y}^i =: res^i$$

  - So raise $\hat{y}^i$ when $res^i > 0$ and lower $\hat{y}^i$ when $res^i < 0$
  - I.e., always try to reduce the absolute value of $res^i$ (obviously)
  - The loss can be improved more when the residual is bigger
    $\rightarrow$so work on fixing the biggest residuals

- Gradient tree boosting, loosely explained:
  - Fit a tree, $\hat{f}_{\text{tree},1}(x^i)$
  - Look at the residuals (prediction errors), $y^i - \hat{f}_{\text{tree},1}(x^i)$
  - Fit a tree to these residuals
  - New $\hat{f}$ = original tree + tree for the residuals
  - New $\hat{f}$ has smaller residuals
  - Repeat

# Gradient Tree Boosting, continued

- How to do gradient tree boosting in more detail:
  - Start with initial tree ("tree number 0"), e.g., $\hat{f}_{\text{tree},0} \equiv 0$
    - so initial residuals are $res_0^i = y^i - 0 = y^i$
  - For each $b = 1, ..., B$
    - Fit a regression tree, $\hat{f}_{\text{tree},b}$, to $(x, res_{b-1})$
    - Shrink tree toward zero, $\nu \hat{f}_{\text{tree},b}$, using shrinkage factor $\nu \in [0, 1]$
    - Update the residuals to, $res_b^i = res_{b-1}^i - \nu \hat{f}_{\text{tree},b}(x^i)$
  - Final prediction is the sum of the resulting $B$ shrunken trees

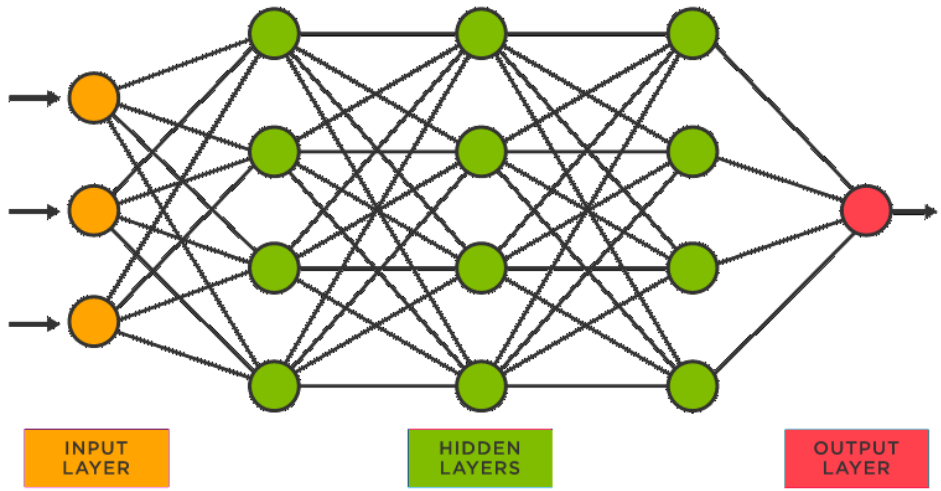
$$\hat{f}_{\text{boosted tree}}(x) = f_{\text{tree},0}(x) + \nu \sum_{b>0} \hat{f}_{\text{tree},b}(x)$$

  - Tuning: Boosting steps $B$, shrinkage factor $\nu$, and tree depth $d$
    - Note: a random forest is an *average* of trees, boosting is a *sum*
    - In random forest, we can let $B \to \infty$ to reduce variance
    - With boosting, large $B$ leads to over-fitting, so must to chosen well
- Popular implementation (more sophisticated): XGBoost
  - Open-source software library for R, Python, Julia, etc.
  - Algorithm for some winning teams of machine learning competitions

# Specific ML Models: Neural Networks

# Neural Networks

- Want to learn $f$ such that $y^i = f(x^i) + \varepsilon^i$
- Use functions of functions of functions: $\hat{f} = k(g(...), ..., g(...))$




- Elements of neural network: overview
  - Input layer (=layer 0): each "neuron" $k$ is just the $k$'th feature, $a_k^{(0)} = x^{i,k}$
  - Hidden layer $h = 1, ..., H$: Each neuron $a_j^{(h)}$ is a function of prior neurons
  - Output layer: our final function is linear in final neurons:

$$\hat{f}(x^i) = \theta_0^{(H)} + \sum_i \theta_i^{(H)} a_i^{(H)}$$

# Neural Networks, continued

- Elements of neural network: more details
  - Input layer (=layer 0): each "neuron" $k$ is just the $k$'th feature, $a_k^{(0)} = x^{i,k}$
  - Hidden layer $h = 1, ..., H$: Each neuron $j$ is a function of prior neurons:

$$a_j^{(h)} = g\left(\theta_{j,0}^{(h-1)} + \sum_i \theta_{j,i}^{(h-1)} a_i^{(h-1)}\right)$$

  - Output layer: $\hat{f}(x^i) = \theta_0^{(H)} + \sum_i \theta_i^{(H)} a_i^{(H)}$
- Comments:
  - Each neuron: first compute a linear function of past neurons
    - with weights given by the $\theta$'s
  - Then transformed with non-linear function $g$
    - Note that $g$ is just a scalar function $\mathbb{R} \to \mathbb{R}$
    - Typical: $g(z) = ReLU(z) = z1_{\{z>0\}}$ or sigmoid $g(z) = \frac{1}{1+e^{-z}}$
  - Lots of hyper-parameters and parameters
    - Hyper: "Design" (number of layers/neurons), $g$ function
    - Parameters: $\theta$'s
    - Typically requires a lot of data!
  - Can be difficult to choose good design and estimate it well

# Feature Importance

# Feature Importance

- The ML literature typically focuses on prediction
  - But model interpretation is also crucial
  - I.e., what is the importance of each feature?
- Measures of features importance
  - Model-specific methods: tailored to a ML method
  - Model-agnostic methods: can be used for any ML method
- Note: substitution effects matter
  - For example, book-to-market will be less important if we include 10 similar value characteristics
  - So it can make sense to aggregate feature importance for highly correlated features
- See also book on "Interpretable ML":

  https://christophm.github.io/interpretable-ml-book/

# Feature Importance: Model-Specific Methods

- OLS/Ridge/Lasso regression
  - When features are on the same scale (i.e., standardized):
    - Importance of feature $j$ is $|\beta^j|$
  - otherwise
    - Importance of feature $j$ is $\sigma(x^j)|\beta^j|$
  - Problematic with high multi-collinearity (especially OLS)

- Regression tree, random forest, or boosting
  - Simple measures of feature importance:
    - Number of times a feature is picked as splitting variable
    - Average drop in prediction error when splitting on variable
  - More sophisticated measure for random forests:
    - Increase in prediction error on "out-of-bag" samples after permuting feature
    - Similar to "permutation feature importance" discussed next

# Model-Agnostic Methods: Permutation Feature Importance

Permutation feature importance

- Pick loss function, $L(y, \hat{f}(x))$, e.g., squared prediction error or $R^2$
- Estimate actual model error

$$l_{\text{orig}} = L(y, \hat{f}(x))$$

- For each feature $j \in \{1, \ldots, K\}$:
  1. Generate $x_{\text{perm},j}$ by permuting (shuffling) feature $j$ randomly
  2. Estimate error:

$$l_{\text{perm},j} = L(y, \hat{f}(x_{\text{perm},j}))$$

  3. Feature importance:

$$FI_j = l_{\text{perm},j} - l_{orig}$$

# Model-Agnostic Methods: Surrogate Model

Surrogate model

- Start with prediction, $\hat{f}(x)$, from a hard-to-interpret model such as a neural network
- Fit interpretable surrogate model, $\hat{f}^{\text{surrogate}}$, to prediction

$$\hat{f}(x^i) = \hat{f}^{\text{surrogate}}(x^i) + \varepsilon^i$$

- E.g., fit regression tree or OLS regression:

$$\hat{f}(x^i) = x^i\beta + \varepsilon^i$$

- Use standard metrics such as $R^2$ to ensure appropriate fit
- Interpret surrogate model

# Model-Agnostic Methods: Shapley Values

## Shapley values

- Method from game theory to allocate reward fairly between players
  - In ML, the features are the players and the reward could be $R^2$
- Notation:
  - $SV_j$ is the Shapley value (importance) of feature $j$
  - $C$ is a coalition of features
  - $C \cup \{j\}$ is the coalition with $j$ added
  - $|C|$ is the number of features in the coalition
  - $K$ is number of features, $\vec{K} = \{1, ..., K\}$, and $\vec{K} \setminus \{j\}$ is the set excluding $j$
  - $V(C)$ is the value function that determines the "worth" of coalition $C$, e.g., the $R^2$ of a model that uses the coalition features
- The Shapley value for feature $j$ is essentially the avg. marginal contribution of $j$ when added to a coalition

$$SV_j = \frac{1}{K} \sum_{C \subseteq \vec{K} \setminus \{j\}} \binom{K-1}{|C|}^{-1} (V(C \cup \{j\}) - V(C)),$$

- SV calculation is computationally demanding
  - Feasible with OLS with a limited set of features
  - But even with OLS the computational burden of SV could be too large
  - Recent advances in ML makes calculating approximate SV feasible for complex models with many features (Lundberg and Lee, 2017)
  - For tree-based model, efficient methods exist (Lundberg et al., 2020)

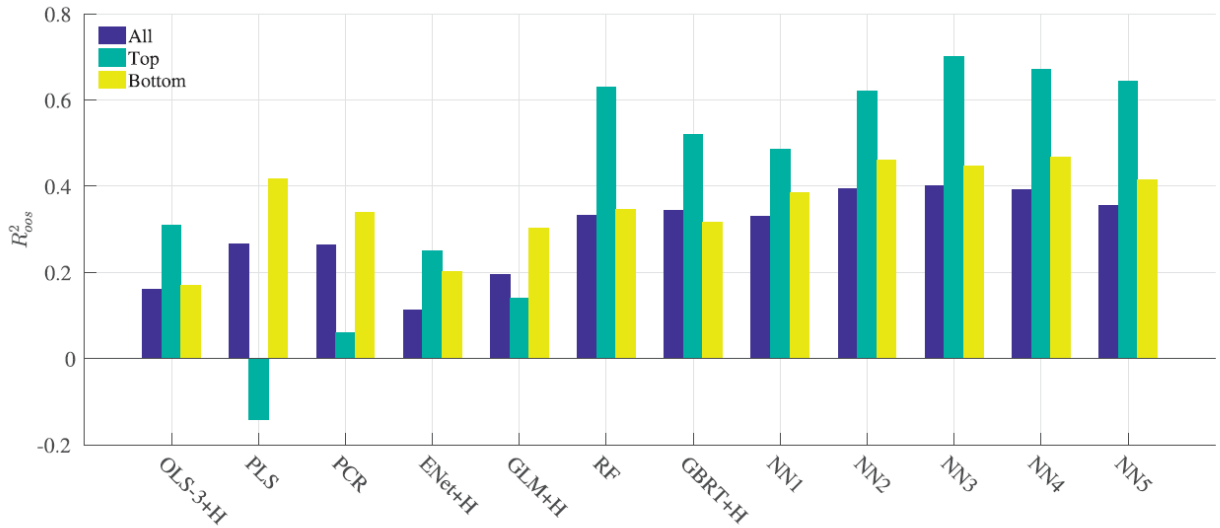
# Applications to Asset Pricing

# Predicting Stock Returns with ML

From: Gu et al. (2020)

**Table 1**
**Monthly out-of-sample stock-level prediction performance (percentage $R^2_{oos}$)**

| | OLS +H | OLS-3 +H | PLS | PCR | ENet +H | GLM +H | RF | GBRT +H | NN1 | NN2 | NN3 | NN4 | NN5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| All | −3.46 | 0.16 | 0.27 | 0.26 | 0.11 | 0.19 | 0.33 | 0.34 | 0.33 | 0.39 | 0.40 | 0.39 | 0.36 |
| Top 1,000 | −11.28 | 0.31 | −0.14 | 0.06 | 0.25 | 0.14 | 0.63 | 0.52 | 0.49 | 0.62 | 0.70 | 0.67 | 0.64 |
| Bottom 1,000 | −1.30 | 0.17 | 0.42 | 0.34 | 0.20 | 0.30 | 0.35 | 0.32 | 0.38 | 0.46 | 0.45 | 0.47 | 0.42 |

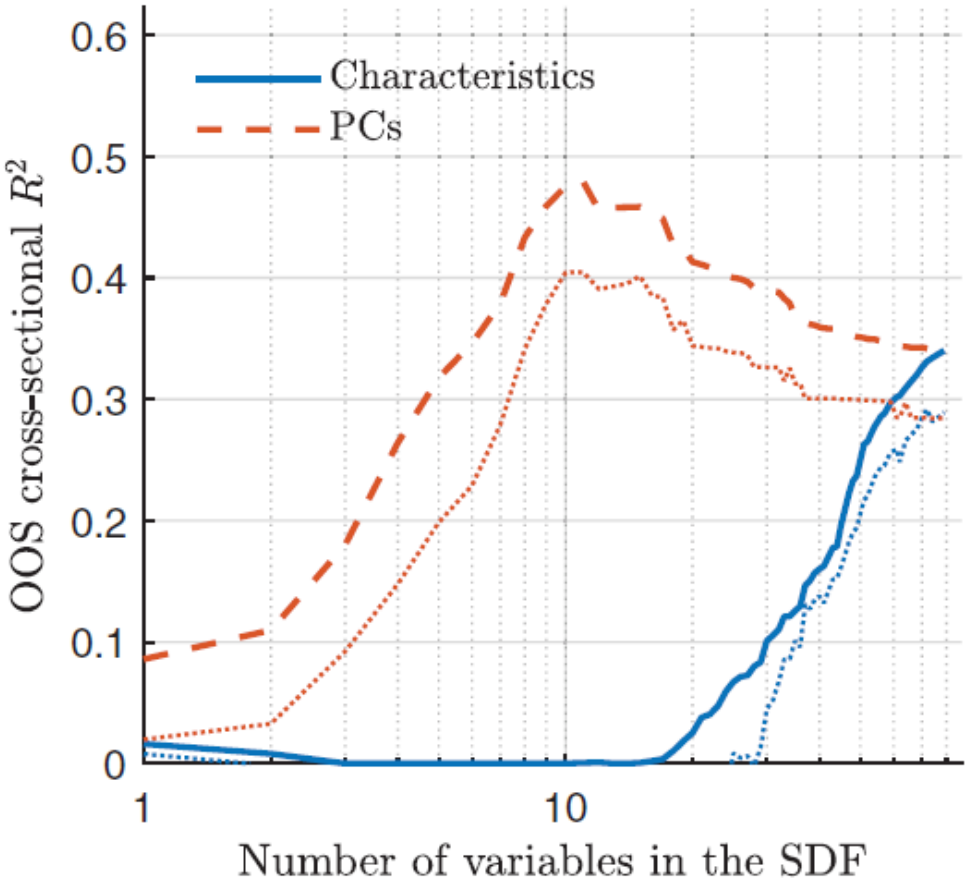



In this table, we report monthly $R^2_{oos}$ for the entire panel of stocks using OLS with all variables (OLS), OLS using only size, book-to-market, and momentum (OLS-3), PLS, PCR, elastic net (ENet), generalize linear model (GLM), random forest (RF), gradient boosted regression trees (GBRT), and neural networks with 1 to 5 layers (NN1–NN5). "+H" indicates the use of Huber loss instead of the $l_2$ loss. We also report these $R^2_{oos}$ within subsamples that include only the top-1,000 stocks or bottom-1,000 stocks by market value. The lower panel provides a visual comparison of the $R^2_{oos}$ statistics in the table (omitting OLS because of its large negative values).

$$R^2_{oos} = 1 - \frac{\sum_{(i,t)\in\mathcal{T}_3}(r_{i,t+1} - \widehat{r}_{i,t+1})^2}{\sum_{(i,t)\in\mathcal{T}_3} r^2_{i,t+1}}$$

# Number of Variables in SDF

From: Kozak et al. (2020), Fig. 6




(WFR portfolios) The maximum OOS cross-sectional R2 attained by a model with n factors (on the x -axis) across all possible values of the prior root expected $SR^2$ ($\kappa$) for models based on original characteristics portfolios (solid) and PCs (dashed). Dotted lines depict $-1$ s.e. bounds of the CV estimator. The sample is daily from September 1964 to December 2017.

# Other Topics in ML

# Other Topics in ML

- Standard errors for predictions
- How to make ML aware of trading costs
  - New paper by Jensen et al. (2022) covered in the last lecture
- Reinforcement learning
  - Designed to handle complex dynamic problems, sometimes called "approximate dynamic programming"
  - Super-human performance on several games such as chess and Go
  - Standard reference: Barto and Sutton (2018), freely available from `http://www.incompleteideas.net/book/the-book-2nd.html`
- Natural language processing (NLP) and textual analysis
- Large language models (LLM)

# References Cited in Slides


Efron, B. and T. Hastie (2021). *Computer Age Statistical Inference, Student Edition: Algorithms, Evidence, and Data Science*, Volume 6. Cambridge University Press.

Gu, S., B. Kelly, and D. Xiu (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies 33*(5), 2223–2273.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

Jensen, T. I., B. T. Kelly, S. Malamud, and L. H. Pedersen (2022). Machine learning about optimal portfolios. *Working paper, Copenhagen Business School*.

Kozak, S., S. Nagel, and S. Santosh (2020). Shrinking the cross-section. *Journal of Financial Economics 135*(2), 271–292.

Lundberg, S. M., G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence 2*(1), 2522–5839.

Lundberg, S. M. and S.-i. Lee (2017). A Unified Approach to Interpreting Model Predictions. In *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*, Number Section 2, pp. 1–10.