The Virtues of Complex Models

Zichao Yang

Zhongnan University of Economics & Law

Date: March 19, 2024

Roadmap

In this chapter, authors challenge the traditional wisdom in econometrics: principle of parsimony.

- Tools for Analyzing Machine Learning Models
- Bigger is Often Better
- The Complexity Wedge

Introduction

One basic idea in model building is **parsimony**.

It is important, in practice, that we employ the smallest possible number of parameters for adequate representations.

— Box, George EP, et al. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

Introduction

Recent results in various non-financial domains contradict this view (i.e., Computer Vision, NPL).

Belkin (2021) concludes "*it appears that the largest technologically feasible networks are consistently preferable for best performance.*"

Evidently, modern machine learning has turned the principle of parsimony on its head.

A Simple Model

Suppose the true DGP for the asset return R follows this form:

$$R_{t+1} = f(X_t) + \epsilon_{t+1}$$

where the set of predictor variables X may be known to the analyst, but the true prediction function f is unknown.

Motivated by **universal approximation theory**, the analyst decides to approximate f with a basic neural network:

$$f(X_t) \approx \sum_{i=1}^{P} S_{i,t}\beta_i$$

A Simple Model

Each feature in this regression is a pre-determined nonlinear transformation of raw predictors:

$$S_{i,t} = \tilde{f}(\omega_t' X_t)$$

Ultimately, the analyst estimates the approximating regression:

$$R_{t+1} = \sum_{i=1}^{P} S_{i,t}\beta_i + \tilde{\epsilon}_{t+1} = \sum_{i=1}^{P} \tilde{f}(\omega_t' X_t)\beta_i + \tilde{\epsilon}_{t+1}$$

And the analyst has T training observations to learn from and she must choose how complex the model should be, the complexity is denoted by P.

Extra: Universal Approximation Theory

Cybenko(1989) shows that arbitrary width neural networks can approximate any continuous function given the use of **sigmoid** activation functions.

Hornik et al. (1989) demonstrate that **multilayer feed-forward** neural networks with just one hidden layer can approximate any continuous function, making them **universal approximators**.

Hornik (1991) further shows that it is not the specific choice of the activation function but rather the **multilayer feed-forward architecture** itself that gives neural networks the potential of being universal approximators.

Leshno et al. (1993) show that multilayer feedforward networks can approximate any continuous function to any degree of accuracy, provided the **activation functions are not polynomial**.

Extra: Universal Approximation Theory

The universal approximation theory means that a feedforward network with a single hidden layer containing a finite number of neurons can approximate **any continuous function** on compact subsets of $\mathcal{R}^n$, given sufficient complexity (i.e, enough neurons) and appropriate weights.

This means that such a network can, in theory, learn any mapping from inputs to outputs, provided the function is continuous and the network has enough capacity.

This theory provides a theoretical justification for using neural networks for a wide range of approximation tasks, from simple function approximation to complex decision making and pattern recognition.

How to choose the P?

Let's first consider the OLS estimator, $\hat{\beta}$.

As $P \to T$, the model begins to fit the data with zero error. The denominator of the least-squares estimator approaches the singularity, which produces explosive variance of $\hat{\beta}$. Usually we say the model is overfitting and does not generalize out-of-sample.

When P moves beyond T, the least-squares problem has multiple solutions. If we invoke the Moore-Penrose pseudo-inverse, the solution is equivalent to the ridge estimator as the shrinkage parameter approaches zero:

$$\hat{\beta}(0^+) = \lim_{z \to 0_+} \left(zI + T^{-1} \sum_t S_t S_t' \right)^{-1} \frac{1}{T} \sum_t S_t R_{t+1}$$

How to choose the P?

The solution $\hat{\beta}(0^{+})$ is often referred to as the "ridgeless" regression estimator.

When $P < T$, OLS is the ridgeless estimator.

At $P = T$, there is still a unique least-squares solution, but the model can exactly fit the training data (for this reason, $P = T$ is called the **interpolation boundary**).

When $P > T$, the ridgeless estimator is one of many solutions that exactly fit the training data, but among these, there is the only solution that achieves the minimum l_2 norm $\hat{\beta}(z)$.

How to choose the P?

Simple models with small P enjoy low variance, but complex models with large P (perhaps even $P > T$) can better approximate the truth (small bias).

What level of model complexity (which P) should the analyst opt for?

Spoiler alert: Authors show that the analyst should use the largest approximating model that she can compute!

Expected out-of-sample forecasting and portfolio performance are increasing in model complexity when appropriate shrinkage is applied.

Shrinkage Methods

In the next subsection, the authors discuss the use of ridge regression to manage complex models.

This serves as a cue for us to explore popular shrinkage methods in machine learning.

We will discuss the following methods:

- Ridge Regression
- Lasso Regression
- Elastic-net Penalty Regression

Ridge Regression

Ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

Here $\lambda \geq 0$ and controls the amount of shrinkage: the larger the λ is, the greater the amount of shrinkage. An equivalent way to write the ridge problem:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2,$$

$$\text{s.t.} \sum_{j=1}^{p} \beta_j^2 \leq t$$

There is a one-to-one correspondence between λ and t.

Ridge Regression

The residual sum-of-squares of the ridge regression is:

$$\mathrm{RSS} = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\intercal(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^\intercal\boldsymbol{\beta}$$

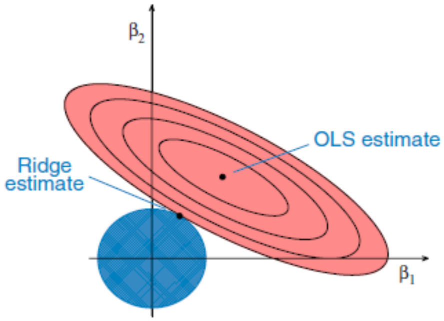
Then the ridge regression solutions are:

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = (\boldsymbol{X}^\intercal\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\intercal\boldsymbol{y}$$

The solution adds a positive constant to the diagonal of $\boldsymbol{X}^\intercal\boldsymbol{X}$ and makes the problem non-singular, even if $\boldsymbol{X}^\intercal\boldsymbol{X}$ is not of full rank. This was the main motivation for ridge regression was first introduced by Hoerl and Kennard (1970).

In the case of orthonormal inputs, the ridge estimates are just a scaled version of the OLS estimates: $\hat{\boldsymbol{\beta}}^{\text{ridge}} = \hat{\boldsymbol{\beta}}/(1+\lambda)$.

Geometric Interpretation of Ridge Regression



Source: PennState STAT 897D

The ellipses correspond to the contours of residual sum of squares (RSS): the inner ellipse has smaller RSS, and RSS is minimized at ordinal least square (OLS) estimates.

We are trying to minimize the ellipse size and circle simultanously in the ridge regression. The ridge estimate is given by the point at which the ellipse and the circle touch.

Ridge Regression: Standardization

In Ridge Regressions, we need to standardize our inputs because of the following reasons:

- **Scaling Coefficients Equally:** Ridge regression penalizes the sum of squared coefficients. If the variables have different scales, those with larger scales will contribute more to the penalty term.
- **Multicollinearity Mitigation:** Standardizing inputs helps alleviate multicollinearity issues, as the regularization penalty is then applied uniformly to all variables, preventing any single variable from dominating the regression.
- **Algorithm Convergence:** Variables with large scales can cause numerical instability and slow convergence, and standardization helps mitigate these issues and improve model performance.

Ridge Regression: How to do Standardization?

The **standardization** process includes two parts:

- **Centering:** subtract the mean from each variable, resulting in centered data where the mean of each variable is zero.
- **Scaling:** divide variables by their standard deviations, which makes them unitless and bring them to a comparable scale.

$$\boldsymbol{X_i} = \frac{\boldsymbol{X_i} - \mu_i}{\sigma_i}$$

After doing the standardization, the β_0 in original equation actually is no longer needed. We can think that $\beta_0 = \bar{\boldsymbol{y}} = \frac{1}{N}\sum_1^N y_i$, and it has been removed in the standardization procedure.

Ridge Regression: How to do Standardization?

Q: The **standardization** process should be applied on:

(1) the whole dataset

(2) the whole training set (3) the whole test set

(4) only X in the training set (5) only X in the test set

(6) only y in the training set (7) only y in the test set

Ridge Regression: How to do Standardization?

None of them is 100% correct. (Yup, I deceived you!)

Here is the recommended procedure of standardization:

- **Fit the scaler on the training set:** Calculate the mean and standard deviation of the features (only X, not y) in the **training set**.
- **Transform the training set:** Standardize the features in the training set using the calculated mean and standard deviation.
- **Transform the test set:** Use the same scaler to standardize the features in the **test set**.

Let's try it out in python!

Ridge Regression: How to Pick the λ?

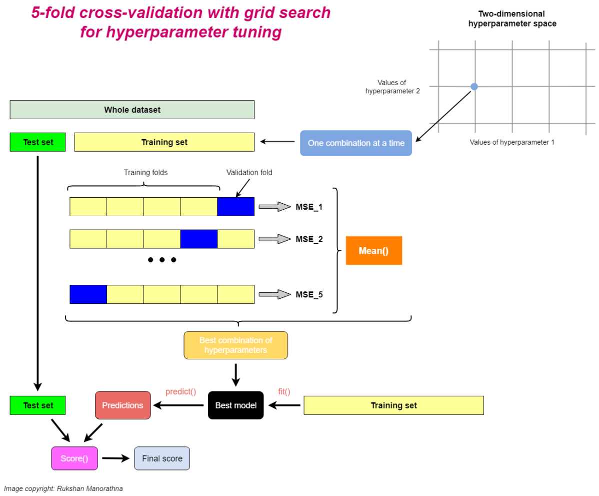
There are two ways to pick the regularization parameter, λ:

1. **AIC, BIC:** A traditional approach would be to choose λ such that some information criterion, e.g., AIC or BIC, is the smallest.

2. **Cross-validation:** A more machine learning-like approach is to perform cross-validation and select the value of λ that minimizes the cross-validated sum of squared residuals (or some other measure). To choose λ through cross-validation, you should choose a set of P different values of λ to test, split the dataset into K folds.

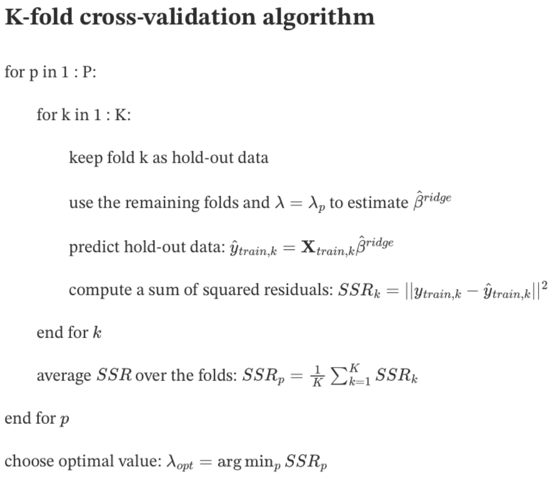
The first approach emphasizes the model's fit to the data, while the latter one is more focused on its predictive performance.

Ridge Regression: How to Pick the λ?


5-fold cross-validation with grid search for hyperparameter tuning
Two-dimensional hyperparameter space
Values of hyperparameter 2
Values of hyperparameter 1
Whole dataset
Test set
Training set
One combination at a time
Training folds
Validation fold
MSE_1
MSE_2
MSE_5
Mean()
Best combination of hyperparameters
Test set
Predictions
predict()
Best model
fit()
Training set
Score()
Final score
Image copyright: Rukshan Manorathna

Source: k-fold cross-validation explained in plain English

Ridge Regression: How to Pick the λ?



K-fold cross-validation algorithm

```
for p in 1 : P:
    for k in 1 : K:
        keep fold k as hold-out data
        use the remaining folds and $\lambda = \lambda_p$ to estimate $\hat{\beta}^{ridge}$
        predict hold-out data: $\hat{y}_{train,k} = \mathbf{X}_{train,k}\hat{\beta}^{ridge}$
        compute a sum of squared residuals: $SSR_k = ||y_{train,k} - \hat{y}_{train,k}||^2$
    end for $k$
    average $SSR$ over the folds: $SSR_p = \frac{1}{K}\sum_{k=1}^{K} SSR_k$
end for $p$
choose optimal value: $\lambda_{opt} = \arg\min_p SSR_p$
```

Let's try it out in python!

Lasso Regression

Similar to the Ridge Regression, the Lasso can be written as:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Equivalent to:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2,$$

$$\text{s.t.} \sum_{j=1}^{p} |\beta_j| \leq t$$

In lasso regression, the penality $\sum_{j=1}^{P} |\beta_j|$ makes the solutions nonlinear in the y_i, and there is no closed form expression as in ridge regression.

Lasso VS. Ridge

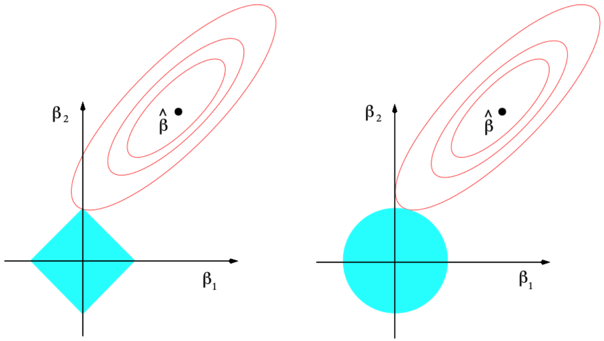


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Source: Hastie, Trevor, et al. *The elements of statistical learning: data mining, inference, and prediction.* New York: springer, 2009.

Now let's try to apply the lasso regression to the Boston Housing Data.

Lasso VS. Ridge

- **Feature Selection:** Lasso translates each coefficient by a constant factor λ, truncating at 0. It can automatically select a subset of relevant features, effectively performing feature selection during the modeling process. Ridge does a proportional shrinkage. It reduces the impact of less important features, but it generally keeps all features in the model.
- **Handling Multicollinearity:** Lasso naturally selects one variable among highly correlated variables and sets the coefficients of the rest to zero. Ridge regression tends to handle multicollinearity by reducing the impact of correlated variables.
- **Solution Stability:** Ridge regression tends to have more stable solutions in the presence of multicollinearity.

Beyond Ridge and Lasso

We can generalize ridge and lasso regression, and view them as Bayes estimates. Consider the criterion:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}, \quad q \geq 0$$

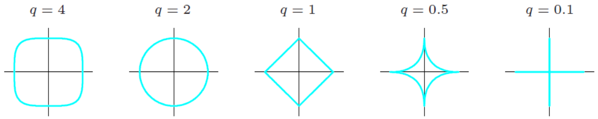


FIGURE 3.12. *Contours of constant value of $\sum_j |\beta_j|^q$ for given values of q.*

Source: Hastie, Trevor, et al. *The elements of statistical learning: data mining, inference, and prediction.* New York: springer, 2009.

The case $q = 1$ (lasso) is the smallest q such that the constraint region is convex; non-convex constraint regions make the optimization problem more difficult.

Elastic-net Penalty Regression

Zou and Hastie (2005) introduced the **elastic-net** penalty:

$$\lambda \sum_{j=1}^{p} (\alpha \beta_j^2 + (1-\alpha)|\beta_j|)$$

The elastic-net selects variables like the lasso, and shrinks together the coefficients of correlated predictors like ridge. It also has considerable computational advantages over the L_q penalties.

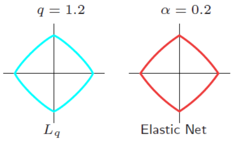


FIGURE 3.13. *Contours of constant value of* $\sum_j |\beta_j|^q$ *for* $q = 1.2$ *(left plot), and the elastic-net penalty* $\sum_j (\alpha\beta_j^2 + (1-\alpha)|\beta_j|)$ *for* $\alpha = 0.2$ *(right plot). Although visually very similar, the elastic-net has sharp (non-differentiable) corners, while the* $q = 1.2$ *penalty does not.*

Source: Hastie, Trevor, et al. *The elements of statistical learning: data mining, inference, and prediction.* New York: springer, 2009.

Shrinkage Methods

Now you understand how to implement these popular shrinkage methods.

Let's return to the paper to explore how these methods can be applied to manage complex models.

Ridge Regression with Generated Features

Let's take a look at the previous equation:

$$R_{t+1} = \sum_{i=1}^{P} S_{i,t}\beta_i + \tilde{\epsilon}_{t+1} = \sum_{i=1}^{P} \tilde{f}(\omega_t' X_t)\beta_i + \tilde{\epsilon}_{t+1}$$

The interpretation of this equation is NOT that asset returns are subject to a large number of linear fundamental driving forces.

It is instead that the *DGP* is unknown, but may be approximated by a nonlinear expansion S of some (potentially small) underlying driving variables, X.

S are "generated features" derived from the raw features X (i.e., via neural network propagation).

Ridge Regression with Generated Features

If we choose the estimator of the above equation using ridge-regularized least squares:

$$\hat{\beta}(z) = \left(zI + T^{-1} \sum_t S_t S_t' \right)^{-1} \frac{1}{T} \sum_t S_t R_{t+1}$$

where z is the ridge shrinkage parameters.

The regularization is critical in complex models. Without regularization, the denominator of the above equation is singular in the high complexity regime ($P > T$).

Random Matrix Theory

The ridge regression formulation above echos machine learning models like neural networks in terms of linear regression.

Here we define another term called **model complexity**: $c = P/T$

Then, the **Random Matrix Theory** can help us to describe the out-of-sample behavior of this model with different complexity:

- If T grows at a faster rate than the number of predictors P, (i.e., $c \to 0$), we can deal it in the traditional econometrics domain. The expected out-of-sample behavior of a model coincides with the behavior estimated in-sample.
- if $P/T \to c > 0$, this corresponds to corresponds to highly parameterized machine learning models with scarce data and is where surprising out-of-sample model behaviors emerge.

Bigger is Often Better

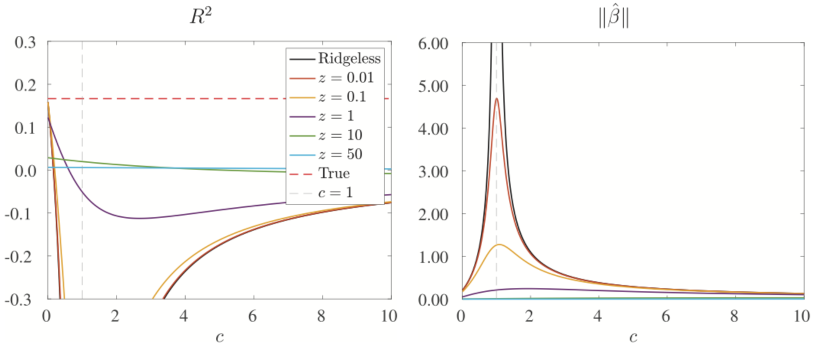


Figure 1. Expected out-of-sample R^2 and norm of least-squares coefficient. This figure shows the limiting out-of-sample R^2 and $\hat{\beta}$ norm as a function of c and z from Proposition 3 assuming Ψ is the identity matrix and $b_* = 0.2$. (Color figure can be viewed at wileyonlinelibrary.com)

Bigger is Often Better

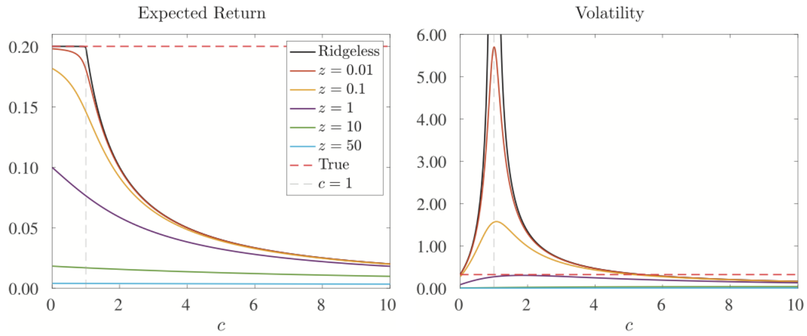


Figure 2. Expected out-of-sample risk and return of market timing. This figure shows the limiting out-of-sample expected return and volatility of the market timing strategy as a function of c and z from Proposition 3 assuming Ψ is the identity matrix and $b_* = 0.2$. (Color figure can be viewed at wileyonlinelibrary.com)

Bigger is Often Better

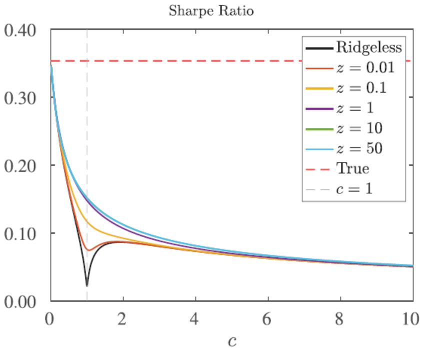


Figure 3. Expected out-of-sample Sharpe ratio of market timing. This figure shows the limiting out-of-sample Sharpe ratio of the market timing strategy as a function of c and z from Proposition 3 assuming Ψ is the identity matrix and $b_* = 0.2$. (Color figure can be viewed at wileyonlinelibrary.com)

Bigger is Often Better

Kelly et al. (2022) conclude that:

"*Our results are not a license to add arbitrary predictors to a model. Instead, we encourage* ***i) including all plausibly relevant predictors*** *and* ***ii)using rich nonlinear models rather than simple linear specifications****. Doing so confers prediction and portfolio benefits, even when training data is scarce, and particularly when accompanied by prudent shrinkage.*"

The Complexity Wedge

Complexity Wedge: the expected difference between in-sample and out-of-sample performance. (Kelly et al., 2022)

In a low complexity environment with $c \approx 0$, the law of large numbers applies, and as a result in-sample estimates converge to the true model.

Because of this convergence, the model's in-sample performance is exactly indicative of its expected out-of-sample performance.

Hence, with no complexity, there is no wedge between in-sample and out-of-sample behavior.

The Complexity Wedge

When $c > 0$, a complexity wedge emerges, consisting of two components:

(1) Complexity inflates the trained model's in-sample predictability relative to the true model's predictability (i.e., overfitting).

(2) High complexity also means that the empirical model does not have enough data (relative to its parameterization) to recover the true model (i.e., limits to learning).

The complexity wedge is the sum of overfitting and limits to learning.

References

- Cybenko, George. *Approximation by superpositions of a sigmoidal function.* Mathematics of control, signals and systems 2.4 (1989): 303-314.
- Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. *Multilayer feedforward networks are universal approximators.* Neural networks 2.5 (1989): 359-366.
- Hornik, Kurt. *Approximation capabilities of multilayer feedforward networks.* Neural networks 4.2 (1991): 251-257.
- Leshno, Moshe, et al. *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.* Neural networks 6.6 (1993): 861-867.
- Kelly, Bryan, Semyon Malamud, and Kangying Zhou. *The virtue of complexity in return prediction.* The Journal of Finance 79.1 (2024): 459-503.