Advanced Empirical Finance: Topics and Data Science

Stefan Voigt

Spring 2024

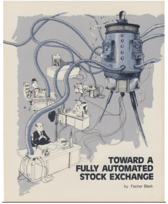
University of Copenhagen and Danish Finance Institute

Machine learning

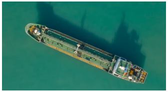
What is Machine learning?

*The definition of "machine learning" is inchoate and is often context specific. We use the term to describe **(i)** a diverse collection of high-dimensional models for statistical prediction, combined with **(ii)** so-called "regularization" methods for model selection and mitigation of overfit and **(iii)** efficient algorithms for searching among a vast number of potential model specifications. (Gu et al. 2020)*

- (i) select between small simplistic and complex ML models
- (i) Focus on predictive accuracy
- (ii) selecting from multiple models in-sample leads to overfitting and poor out-of-sample performance
- (ii) "regularization" methods for model selection
- (iii) challenge in terms of computational effort



What makes ML in Finance special?



Challenges (Israel, Kelly, Moskowitz, 2019)

- Limited data (left-hand side limited by *T*)
- Markets evolve and thus even lower effective sample size
- By market efficiency: small signal-to-noise ratio (limited predictability)
- Data potentially unstructured (company announcements)

But...

- Machine learning methods on their own do not identify fundamental associations among asset prices and conditioning variables

Overview: Empirical Asset Pricing via Machine Learning

- Familiarize yourself with the paper “Empirical Asset Pricing via Machine Learning” by Gu et al. (2020)
- comparative analysis of machine learning methods for the canonical problem of measuring asset risk premiums
- “We demonstrate large economic gains to investors using *machine learning forecasts*, in some cases doubling the performance of leading regression-based strategies from the literature.”

Machine learning roadmap

1. Bias-Variance Trade-off
2. Penalized Linear Regressions (Ridge and Lasso)
3. Regression Trees and Random Forests
4. Neural Networks
5. Advanced case studies and applications

Your task:

- Return prediction for all CRSP-listed stocks
- Large set of macroeconomic predictors
- Hundreds of predictive firm and economic characteristics
- You should study Gu et al. (2020) in depth!
- **Exercises:** Prepare the dataset as explained in Section 2.1 of Gu et al. (2020)

Bias-Variance Trade-off

Unbiased, linear estimators

$$E_t\left(r_{i,t+1}\right) = g(x_{i,t}) \overset{??}{=} \beta' x_{i,t}$$

- Machine learning prescribes a vast collection of high-dimensional models that attempt to predict future quantities of interest while imposing regularization
- We know: OLS is the best linear unbiased estimator (BLUE)
- "Best" = the lowest variance estimator among all other *unbiased linear* estimators
- Requiring the estimator to be *linear* is binding since *nonlinear* estimators exist (e.g., neural networks or regression trees)
- Likewise, *unbiased* is crucial since *biased* estimators do exist

Biased estimators?

- *Shrinkage* methods: the variance of the OLS estimator can be high as OLS coefficients are unregulated
- If judged by Mean Squared Error (MSE), biased estimators could be more attractive if they produce substantially smaller variance than OLS

Shortcomings of OLS

- Let β denote the true regression coefficient and let $\hat{\beta} = (X'X)^{-1} X'y$, where X is a $(T \times N)$ matrix of explanatory variables
- Then, the variance of the (unbiased) OLS estimate $\hat{\beta}$ is given by

$$\begin{aligned} Var\left(\hat{\beta}\right) &= E\left(\left(\hat{\beta} - \beta\right)\left(\hat{\beta} - \beta\right)'\right) \\ &= E\left((X'X)^{-1}X'\varepsilon\varepsilon'X(X'X)^{-1}\right) \\ &= \sigma_\varepsilon^2 E\left((X'X)^{-1}\right) \end{aligned}$$

  where ε is the vector of residuals and σ_ε^2 is the variance of the error term
- When the predictors are highly correlated, the term $(X'X)^{-1}$ quickly explodes
- Even worse: the OLS solution is not unique if X is not of full rank

OLS in a prediction context

1. restrictive
2. may provide poor predictions, may be subject to *over-fitting*
3. does not penalize for model complexity and could be difficult to interpret

The Bias-Variance Trade-off

- Assume the model
$$y = f(x) + \varepsilon, \quad \varepsilon \sim (0, \sigma_\varepsilon^2)$$
- $\hat{\beta}^{ols}$ has a host of well-known properties (Gauss-Markov)
- But: Can we choose $\hat{f}(x)$ to fit future observations well?
- MSE depends on the model as follows:

$$\begin{aligned}
E(\hat{\varepsilon}^2) &= E((y - \hat{f}(\mathbf{x}))^2) = E((f(\mathbf{x}) + \varepsilon - \hat{f}(\mathbf{x}))^2) \\
&= \underbrace{E((f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2)}_{\text{total quadratic error}} + \underbrace{E(\varepsilon^2)}_{\text{irreducible error}} \\
&= E\left(\hat{f}(\mathbf{x})^2\right) + E\left(f(\mathbf{x})^2\right) - 2E\left(f(\mathbf{x})\hat{f}(\mathbf{x})\right) + \sigma_\varepsilon^2 \\
&= E\left(\hat{f}(\mathbf{x})^2\right) + f(\mathbf{x})^2 - 2f(\mathbf{x})E\left(\hat{f}(\mathbf{x})\right) + \sigma_\varepsilon^2 \\
&= \underbrace{\mathrm{Var}\left(\hat{f}(\mathbf{x})\right)}_{\text{variance of model}} + \underbrace{E\left((f(\mathbf{x}) - \hat{f}(\mathbf{x}))\right)^2}_{\text{squared bias}} + \sigma_\varepsilon^2
\end{aligned}$$

- A biased estimator with small variance may have a lower MSE than an unbiased estimator

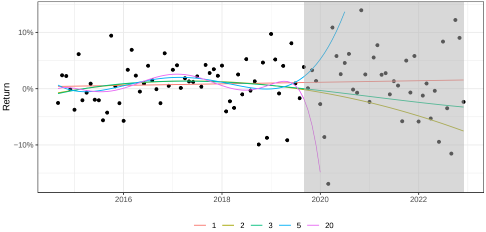
Over-fitting example

- 100 monthly manufacturing industry excess returns
- Estimate a polynomial regression

$$r_t = \alpha + \sum_{p=1}^{P} \beta_p t^p$$

where t is a time index, ranging from 1 to 60

- Evaluate the performance in-sample and out-of-sample for $P = 1, 2, 3, 5, 20$



Ridge Regression

- Introduced by Hoerl and Kennard (1970a, 1970b)
- Impose a penalty on the L_2 norm of the parameters $\hat{\beta}$ such that for $c \geq 0$ the estimation takes the form

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} (y - X\beta)' (y - X\beta) \;\text{ s.t. }\; \beta'\beta \leq c$$

- Standard optimization procedure yields

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

- Hyper parameter λ (c) controls the amount of regularization
- Note that $\hat{\beta}^{\text{ridge}} = \hat{\beta}^{\text{ols}}$ for $\lambda = 0$ ($c \rightarrow \infty$) and $\hat{\beta}^{\text{ridge}} \rightarrow 0$ for $\lambda \rightarrow \infty$ ($c \rightarrow 0$)
- $(X'X + \lambda I)$ is non-singular even if $X'X$ is
- *Note:* Usually, the intercept is not penalized (in practice: demean y)

Ridge Regression

- Let $D := X'X$

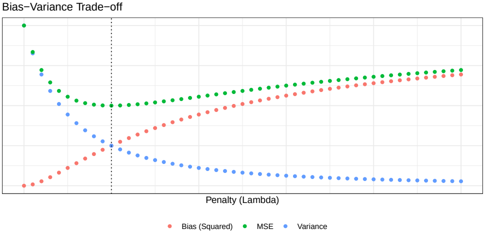
$$\begin{aligned}\hat{\beta}^{\text{ridge}} &= (X'X + \lambda I)^{-1} X'y \\ &= (D + \lambda I)^{-1} DD^{-1} X'y \\ &= \left(D\left(I + \lambda D^{-1}\right)\right)^{-1} D\hat{\beta}^{\text{ols}} \\ &= \left(I + \lambda D^{-1}\right)^{-1} D^{-1} D\hat{\beta}^{\text{ols}} = (I + \lambda D)^{-1} \hat{\beta}^{\text{ols}}\end{aligned}$$

- $\hat{\beta}^{\text{ridge}}$ is biased because $E(\hat{\beta}^{\text{ridge}} - \beta) \neq 0$ for $\lambda \neq 0$
- *But* at the same time (under homoscedastic error terms)

$$\text{Var}(\hat{\beta}^{\text{ridge}}) = \sigma_{\varepsilon}^{2} (D + \lambda I)^{-1} X'X (D + \lambda I)^{-1}$$

- You can show that $\text{Var}(\hat{\beta}^{\text{ridge}}) \leq \text{Var}(\hat{\beta}^{\text{ols}})$
- Trade-off between bias and variance of the estimator!

Bias variance trade-off with the ridge trace



A case study

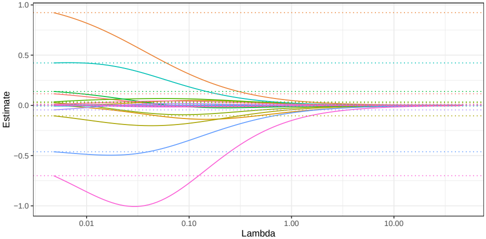
- Data for case study: macroeconomic predictors from for the paper "A Comprehensive Look at The Empirical Performance of Equity Premium Prediction" (Goyal, 2008)
- Monthly variables that have been suggested as good predictors for the equity premium: Dividend Price Ratio, Earnings Price Ratio, Stock Variance, Net Equity Expansion, Treasury Bill rate, and inflation
- Monthly Fama-French 3-factor returns (market, small-minus-big, and high-minus-low book-to-market valuation sorts)
- Monthly q-factor returns from Hou, Xue, and Zhang (2015)
- Monthly portfolio returns from 10 different industries according to the definition from Kenneth French's homepage
- The regression specification is

$$\underbrace{r_{i,t}}_{\text{industry } i} = \gamma_{i,0} + \gamma_{i,1} \underbrace{x_{t-1}}_{\text{macro}\times\text{factor}} + \varepsilon_t$$

- Package `glmnet` fits generalized linear models via penalized maximum likelihood
- **Exercise:** Implement Ridge on your own before using `glmnet`

Ridge trace

- Below I visualize the *ridge trace* for different values of the penalty λ
- To keep things simple, I restrict the sample to the manufacturing portfolio



The Lasso (Tibshirani, 1996)

- Obvious “drawback” of Ridge regression: no variable selection
- Instead of proportional shrinkage (Ridge), the Lasso (least absolute shrinkage and selection operator) translates each coefficient by a constant factor λ, truncating at zero
- Lasso implements a L_1 penalization on the parameters s.t. $\|\beta\|_1 := \sum_k |\beta_k| \leq c$

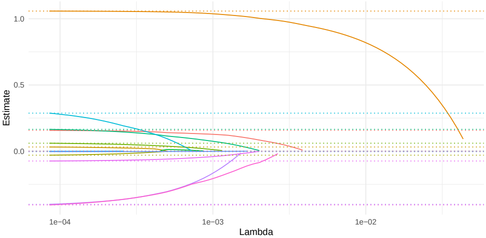
$$\hat{\beta}^{\text{Lasso}} = \arg\min_{\beta} (Y - X\beta)' (Y - X\beta) \text{ s.t. } \sum_{k=1}^{K} |\beta_k| < c(\lambda).$$

- Equivalent optimization problem for a hyperparameter λ:

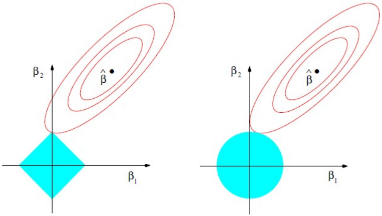
$$\hat{\beta}_{\lambda}^{\text{Lasso}} = \arg\min_{\beta} (Y - X\beta)' (Y - X\beta) + \lambda \sum_{k=1}^{K} |\beta_k|.$$

- No closed-form solution but efficient algorithms (`glmnet`)
- Also here: typically no penalization on the intercept term
- **Exercise:** Implement Lasso estimation on your own before using `glmnet`

Lasso with glmnet



Difference between Ridge and Lasso



- Ridge can be interpreted as a Bayesian posterior mean with a **Normal** prior on β
- Lasso can be interpreted as a Bayesian posterior mean with a **Laplace** prior on β

$$\beta^{\text{Ridge}} \propto \exp\left(-\frac{\lambda\beta^2}{\sigma}\right) \qquad \beta^{\text{Lasso}} \propto \frac{\lambda}{2\sigma}\exp\left(-\frac{\lambda|\beta|}{\sigma}\right)$$

- Next step: Elastic net (Zhou & Hastie, 2005) combines L_1 and L_2 penalization
- Encourages a grouping effect, where strongly correlated predictors tend to be in or out of the model together.

$$\hat{\beta}^{\text{EN}} = \arg\min_{\beta}\,(Y - X\beta)'\,(Y - X\beta) + \lambda(1-\rho)\sum_{k=1}^{K}|\beta_k| + \frac{1}{2}\lambda\rho\sum_{k=1}^{K}\beta_k^2$$

Cross-validation

“Hyperparameter-tuning” with cross-validation

- Goal: find an algorithm that produces predictors $\hat{y}$ for an outcome y that minimizes the mean squared prediction error:

$$\text{MSPE} = \text{E}\left(\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2\right)$$

- We can only estimate the MSPE:

$$\hat{\text{MSPE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2$$

1. Because our data is random, the apparent error is a random variable
2. If we train an algorithm on the same dataset that we use to compute the apparent error, we might be overfitting

Cross-validation

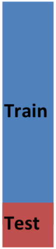
- Cross-validation is a technique that permits us to alleviate both these problems
- Think of the true MSPE as the average of many apparent errors obtained by applying the algorithm to B new random samples of the data, none of them used to train the algorithm

$$\frac{1}{B}\sum_{b=1}^{B}\frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i^b - y_i^b\right)^2$$

- Idea: randomly generate smaller datasets that are not used for training and instead used to estimate the true error
- Recall: The goal is to choose hyperparameters λ to obtain the smallest MSPE

Sample split

- Carve out a piece of our dataset and pretend it is an independent dataset: divide it into a *training set* (blue) and a *test set* (red)
- Train the algorithm exclusively on the training set and use the test set only for evaluation purposes (not for filtering out rows, not for selecting features, nothing!)
- Typical choices are to use 10%-20% of the data for testing



Validation sample

- To choose from the set of hyperparameters, we further divide our training sample without using our test sample!
- For each set of algorithm parameters being considered, we want an estimate of the MSPE, and then we will choose the parameters with the smallest MSE

1. Prespecify a grid of hyperparameters
2. Obtain predictors $\hat{y}_i(\lambda)$ to denote the predictors for the used parameters λ
3. Compute

$$\text{MSPE}(\lambda) = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i^b(\lambda) - y_i^b\right)^2$$

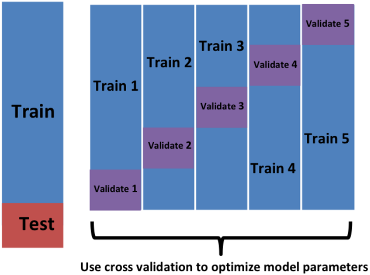
- With K-fold cross-validation, we do it K times: pick a validation set with $M = N/K$ observations at random and think of these as a random sample $y_1^b, \ldots, y_M^b$, with $b = 1$

K-fold cross-validation

- Fit the model in the training set, then compute the apparent error on the independent set

$$\hat{\text{MSPE}}_b(\lambda) = \frac{1}{M}\sum_{i=1}^{M}\left(\hat{y}_i^b(\lambda) - y_i^b\right)^2$$

- Take K samples to reduce the variance of the estimate
- In K-cross validation, we randomly split the observations into K non-overlapping sets:



K-fold cross-validation

- repeat the calculation above for each of these sets $b = 1, \dots, K$ and obtain $\hat{\text{MSE}}_1(\lambda), \dots, \hat{\text{MSPE}}_K(\lambda)$ and compute the average

$$\hat{\text{MSPE}}(\lambda) = \frac{1}{K}\sum_{b=1}^{K} \hat{\text{MSPE}}_b(\lambda)$$

- final step: select the λ that minimizes the MS

Considerations for selecting *K*

- Large values of K are preferable. The training data better imitates the original dataset
- Larger values of K will have much slower computation time
- One way to improve the variance of our final estimate is to take more samples. To do this, pick K sets of some size at random (not necessarily non-overlapping)
- The bootstrap: at each fold, pick observations at random with replacement (which means the same observation can appear twice)

Machine learning in one expression

- Method defines function class F (in this case, linear model) and a regularizer $R(f)$ (shrinkage intensity λ, later: depth of the tree) that expresses the complexity of a function
- Picking the prediction function then involves two steps

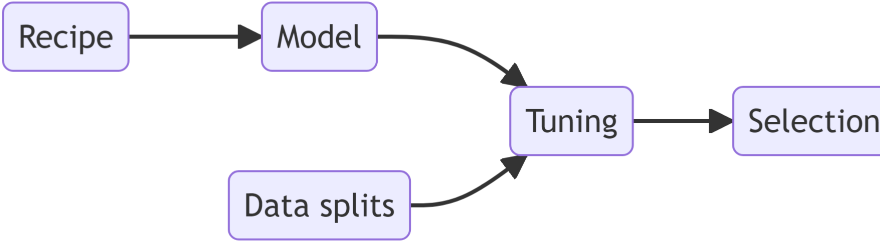
1. conditional on a level of complexity, pick the best in-sample loss-minimizing function

$$\min \sum_{i=1}^{n} L\left(f(x_i), y_i\right) \text{ over } f \in F \text{ subject to } R(f) \leq c$$

2. estimate the optimal level of complexity c using empirical tuning (cross-validation)

Prepare the data

- R and Python provide unparalleled workflows for ML: `tidymodels` and `scikit-learn`



- Preprocessing steps (`recipe`)
- Model definition with declaration of tuning parameters
- Data split handling
- Tuning and model selection

Tidymodels walk-through

```r
library(tidymodels) # For ML applications
library(timetk)
split <- initial_time_split(
  data |>
    filter(industry == "manuf") |>
    select(-industry),
  prop = 4 / 5
)
```

- We start with a pre-processing plan (`recipe`)
- We remove the column *month*, include all interaction terms between factors and macroeconomic predictors, and demean and scale each variable such that the standard deviation is one
- Do not move on if it is unclear why we do not simply use `mutate`!

```r
rec <- recipe(ret_excess ~ ., data = training(split)) |>
  step_rm(month) |> # remove date variable
  step_interact(terms = ~ contains("factor"):contains("macro")) |> # interaction terms
  step_normalize(all_predictors()) |> # scale to unit standard deviation
  step_center(ret_excess , skip = TRUE) # demean variables
```

Build a model with tidymodels

- Makes use of a range of packages combined in `tidymodels`

```r
lm_model <- linear_reg(
  penalty = 0.0001,
  mixture = 1
) |> set_engine("glmnet", intercept = FALSE)
```

- `lm_model` contains the definition of our model with all required information.
- `set_engine("glmnet")` indicates the API character of the `tidymodels` workflow: Under the hood, the package `glmnet` is doing the heavy lifting, while `linear_reg` provides a unified framework to collect the inputs
- Why is this amazing? You can change the model (e.g., change mixture to get Ridge or call neural net instead of linear regression)!

```r
lm_fit <- workflow() |>
  add_recipe(rec) |>
  add_model(lm_model)
```

- `workflow` ends with combining everything necessary for the (serious) data science workflow: a recipe and a model. So now we are ready to use `fit`.

```r
lm_fit |> fit(data = training(split))
```

Tune a model

- Recall: Specify a grid of hyperparameters, obtain predictors $\hat{y}_i(\lambda)$ to denote the predictors for the used parameters λ and compute MSPE(λ)

```r
lm_model <- linear_reg(
  penalty = tune(),
  mixture = tune()
) |> set_engine("glmnet")
lm_fit <- lm_fit |> update_model(lm_model) # Update the existing model
```

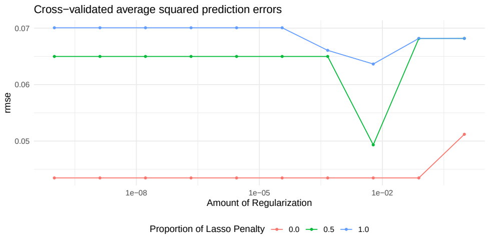
- time-series cross-validation sample: tune with 20 random samples of length five years with a validation period of 4 years

```r
data_folds <- time_series_cv(
  data        = training(split),
  date_var    = month,
  initial     = "5 years",
  assess      = "48 months", cumulative  = FALSE, slice_limit = 20
)
```

```r
lm_tune <- lm_fit |>
  tune_grid(
    resample = data_folds,
    grid = grid_regular(penalty(), mixture(), levels = c(10, 3)),
    metrics = metric_set(rmse)
  )
```

Selecting the best model

```r
autoplot(lm_tune) +
  theme_minimal() +
  theme(legend.position = "bottom") +
  labs(title = "Cross-validated average squared prediction errors")
```



Scikit-learn recipes

- Skicit-learn offers a similar range of preprocessing (`recipe`) steps
- Interaction terms have to be built manually

```
preprocessor = ColumnTransformer(
  transformers=[
    ("scale", StandardScaler(),
    [col for col in data.columns
      if col not in ["ret_excess", "month", "industry"]])
  ],
  remainder="drop",
  verbose_feature_names_out=False
)
```

Scikit-learn model building

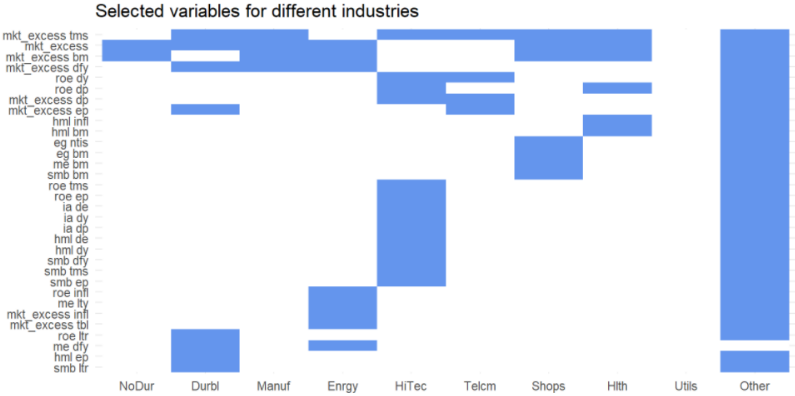
```python
lm_model = ElasticNet(
  alpha=0.007,
  l1_ratio=1,
  max_iter=5000,
  fit_intercept=False
)

lm_pipeline = Pipeline([
  ("preprocessor", preprocessor),
  ("regressor", lm_model)
])


# Easy to fit a model
lm_fit = lm_pipeline.fit(
  data_manufacturing_training,
  data_manufacturing_training.get("ret_excess")
)
```

Scikit-learn model tuning

```
data_folds = TimeSeriesSplit(
  n_splits=n_splits,
  test_size=assessment_months,
  max_train_size=initial_years * length_of_year
)

params = {
  "regressor__alpha": alphas,
  "regressor__l1_ratio": (0.0, 0.5, 1)
}

finder = GridSearchCV(
  lm_pipeline,
  param_grid=params,
  scoring="neg_root_mean_squared_error",
  cv=data_folds
)

finder = finder.fit(
  data_manufacturing, data_manufacturing.get("ret_excess")
)
```

Advanced Tools

- In the exercises, you will go further: Compute lasso (with penalty as tuning parameter) for **all** industries in the sample
- The figure below illustrates for each industry the selected variables: What are your expectations?



Nonlinear methods

Regression Trees

- Regression trees have become a popular machine learning approach for incorporating multiway predictor interactions
- Trees are fully nonparametric and possess a logic that departs markedly from traditional regressions
- Trees are designed to find groups of observations that behave similarly
- A tree “grows” in a sequence of steps
- At each step, a new “branch” sorts the data leftover from the preceding step into bins based on one of the predictor variables
- This sequential branching slices the space of predictors into rectangular partitions and approximates the unknown function $f(x_i)$ with the average value of the outcome variable within each partition

How do regression trees work?

- We partition the predictor space into J non-overlapping regions, $R_1, R_2, \ldots, R_J$
- For any predictor x that falls within region R_j we estimate $f(x)$ with the average of the training observations y_i for which the associated predictor x_i is also in R_j
- Once we select a partition $\mathbf{x}$ to split to create the new partitions, we find a predictor j and value s that define two new partitions, which we will call $R_1(j, s)$ and $R_2(j, s)$, that split our observations in the current partition by asking if x_j is bigger than s:

$$R_1(j, s) = \{\mathbf{x} \mid x_j < s\} \text{ and } R_2(j, s) = \{\mathbf{x} \mid x_j \geq s\}$$

- To pick j and s, we find the pair that minimizes the residual sum of square (RSS):

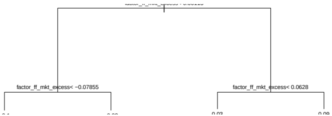
$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

- Note: We do not scale by $\#R_k(j, s)$
- Question: What are the hyperparameter decisions?

Intuition with R

- Also here: In the **exercises**, you will implement a regression tree on your own

```r
library(rpart)
model <- rpart(
  ret_excess ~ .,
  data = training(split) |> select(-month),
  control = rpart.control(maxdepth = 2))

plot(model, compress = TRUE)
text(model, cex = 0.7, fancy = FALSE, all = FALSE)
```



Random forests and Bagging

- Single tree models suffer from high variance
- Pruning the tree helps reduce this variance
- Bootstrap aggregating (Bagging) is one such approach (initially proposed by Breiman, 1996)
- Bagging combines and averages multiple models. Averaging across multiple trees reduces the variability of any one tree and reduces overfitting, which improves predictive performance. Bagging follows three simple steps:

Growing a forest

- Build B decision trees $T_1, \dots, T_B$ using the training sample
- The bootstrap: Create a bootstrap training set by sampling N observations from the training set with replacement
- Also: randomly selecting features to be included in the building of each tree
- For each observation in the test set, form a prediction $\hat{y} = \frac{1}{B} \sum_{i=1}^{B} \hat{y}_{T_i}$

Hyperparameter tuning for regression trees

```r
# install.packages("ranger") # Regression trees in R
rf_model <- rand_forest(
  trees = 50,
  min_n = 20
) |>
  set_engine("ranger") |>
  set_mode("regression")
```

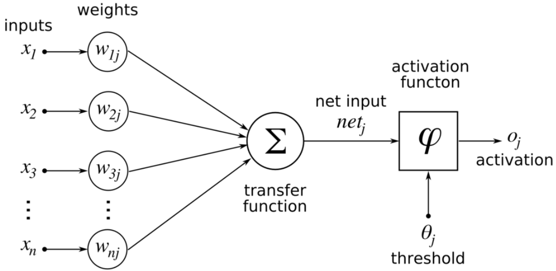
- Fitting the model follows the same convention as for the penalized regressions before

```r
rf_fit <- workflow() |>
  add_recipe(rec) |>
  add_model(rf_model) |>
  fit(data = training(split))
```

Neural networks

Three major takeaways from the biological neuron

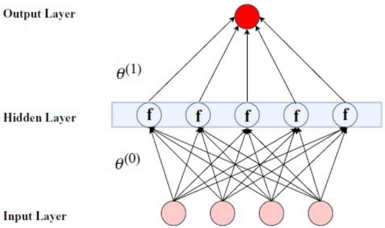
1. The neuron only generates a signal if a sufficient number of input signals enter the neuron's dendrites (all or nothing)
2. Neurons receive inputs from many adjacent neurons upstream and can transmit signals to many adjacent signals downstream (cumulative inputs)
3. Each neuron has its threshold for activation (synaptic weight)



Feed-forward neural networks

- Neural networks have theoretical underpinnings as “universal approximators” for any smooth predictive association (Hornik, 1991)
- their complexity ranks neural networks among the *least transparent, least interpretable, and most highly parameterized* machine learning tools
- consist of an “input layer” of raw predictors, one or more “hidden layers” that interact and nonlinearly transform the predictors, and an “output layer” that aggregates hidden layers into an outcome prediction
- number of units (neurons) in the input layer is equal to the dimension of the predictors
- output layer usually consists of one neuron (for regression) or multiple neurons for classification

Neural networks: Intuition



- 4 input units
- In this example, there is one hidden layer with five neurons
- Each neuron receives information from each input layer
- The results from each neuron, $x_k^1 = f\left(\theta_k^0 + \sum_{j=1}^{4} z_j \theta_{k,j}^0\right)$ are finally aggregated into one output forecast

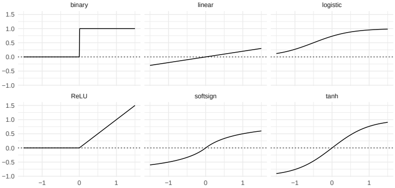
$$\theta_0^1 + \sum_{j=1}^{5} x_j^1 \theta_j^1$$

Activation function

- Each neuron applies a nonlinear "activation function" f to its aggregated signal before sending its output to the next layer

$$x_k^l = f\left(\theta_0^k + \sum_{j=1}^{N^l} z_j \theta_{l,j}^k\right)$$

- Easiest case: $f(x) = \alpha + \beta x$ resembles linear regression
- Typical activation functions are sigmoid ($f(x) = (1 + e^{-x})^{-1}$) or ReLu ($f(x) = max(x, 0)$)



Neural networks: Architecture and Implementation

Plenty of decisions

- Depth (number of hidden layers), Activation function, number of neurons, connections of units (dense or sparse), regularization to avoid overfitting, learning rate
- There is no clear theoretical guidance on these choices but rather a large number of rules of thumbs
- Despite the computational challenges, implementation in R is not tedious at all: We can use the API to tensorflow
- Take a look at this impressive visualization
- Due to the data transformation process that DNNs perform, they are susceptible to the individual scale of the feature values: Standardize the feature sets!

Exercise

- Follow the `Tensorflow` installation steps and study some of their examples

Backpropagation

- How to train the model?
- For given weights w, we run the neural network and receive output $\tilde{y}$ as a function of the input values X (as well as the network architecture and the current weights)
- Compute the loss $L(\tilde{y}, y)$, e.g. MSE
- The aim is to choose w to minimize the Loss
- $\Rightarrow$ Differentiate with respect to w and move in the direction of the negative gradient (step size is a hyperparameter, often called learning rate)

$$\Delta\vec{w} = r \cdot \left(\frac{\partial P}{\partial w_0}, \frac{\partial P}{\partial w_1}, \ldots, \frac{\partial P}{\partial w_q} \right)$$

- For a thorough discussion of backpropagation, consider watching this lecture

Deep neural networks in R

- `keras` provides a helpful interface to create and train a deep neural network
- The example below generates a sequential model with 21 input units, two hidden layers with 64 and 16 neurons, respectively, and ReLu activation functions
- Dropout and L_2 regularization is easy to include
- Output is a single neuron with linear (default) activation function which can be used for predictions

```r
library(keras)
model <- keras_model_sequential() |>
  layer_flatten(input_shape = 21) |>
  layer_dense(units = 64, activation = "relu") |>
  # layer_dropout(0.6) |>
  layer_dense(units = 16,
              activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.001)) |>
  layer_dense(1)
```

Predict returns with neural networks

- All that is needed is to compile and then fit the model

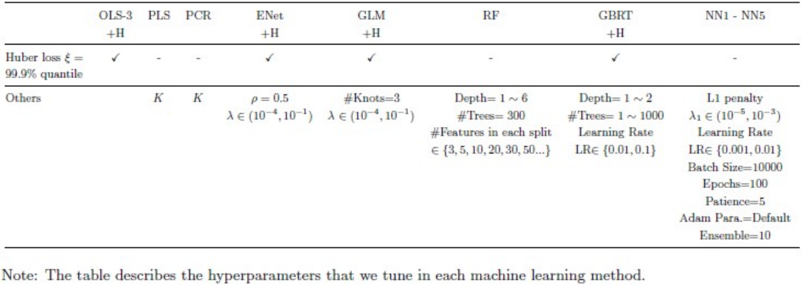
```
model |> compile(loss = "mse",
                  optimizer = optimizer_rmsprop(),
                  metrics = "mean_absolute_error")

model |> fit(x = training(split) |> select(-month, -ret),
              y = training(split) |> pull(ret),
              validation_data = list(testing(split) |> select(-month, -ret),
                                     testing(split) |> pull(ret)),
    epochs = 600)
```

Empirical Asset Pricing via Machine Learning

- Now you are equipped with everything needed to follow Gu et al. (2020): Methods, Data, Procedure

Hyperparameter tuning

- Consult the Online Appendix

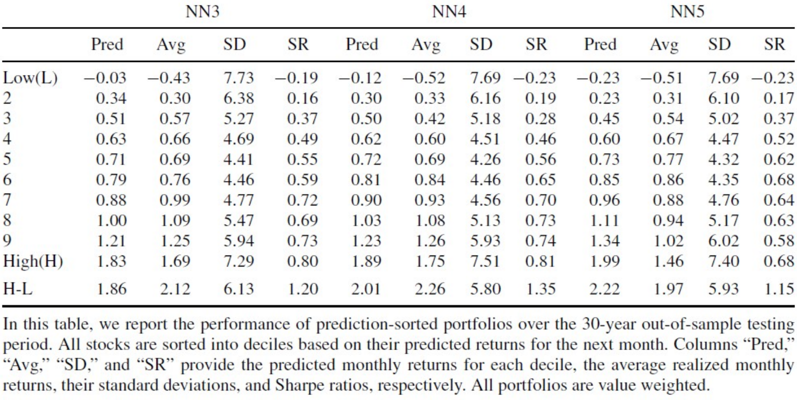


	OLS-3 +H	PLS	PCR	ENet +H	GLM +H	RF	GBRT +H	NN1 - NN5
Huber loss $\xi =$ 99.9% quantile	✓	-	-	✓	✓	-	✓	-
Others		K	K	$\rho = 0.5$ $\lambda \in (10^{-4}, 10^{-1})$	#Knots=3 $\lambda \in (10^{-4}, 10^{-1})$	Depth= $1 \sim 6$ #Trees= 300 #Features in each split $\in \{3, 5, 10, 20, 30, 50...\}$	Depth= $1 \sim 2$ #Trees= $1 \sim 1000$ Learning Rate LR$\in \{0.01, 0.1\}$	L1 penalty $\lambda_1 \in (10^{-5}, 10^{-3})$ Learning Rate LR$\in \{0.001, 0.01\}$ Batch Size=10000 Epochs=100 Patience=5 Adam Para.=Default Ensemble=10

Note: The table describes the hyperparameters that we tune in each machine learning method.

Results

- Portfolio sorts based on return prediction: “At the end of month *t*, we calculate one-month-ahead out-of-sample stock return predictions for each method. We then sort stocks into deciles based on each model’s forecasts. We buy the highest expected return stocks (decile 10) and sell the lowest (decile 1). At the end of the month *t* + 1, we can calculate the realized returns of the portfolios (buy side and sell side respectively)”



	NN3				NN4				NN5			
	Pred	Avg	SD	SR	Pred	Avg	SD	SR	Pred	Avg	SD	SR
Low(L)	−0.03	−0.43	7.73	−0.19	−0.12	−0.52	7.69	−0.23	−0.23	−0.51	7.69	−0.23
2	0.34	0.30	6.38	0.16	0.30	0.33	6.16	0.19	0.23	0.31	6.10	0.17
3	0.51	0.57	5.27	0.37	0.50	0.42	5.18	0.28	0.45	0.54	5.02	0.37
4	0.63	0.66	4.69	0.49	0.62	0.60	4.51	0.46	0.60	0.67	4.47	0.52
5	0.71	0.69	4.41	0.55	0.72	0.69	4.26	0.56	0.73	0.77	4.32	0.62
6	0.79	0.76	4.46	0.59	0.81	0.84	4.46	0.65	0.85	0.86	4.35	0.68
7	0.88	0.99	4.77	0.72	0.90	0.93	4.56	0.70	0.96	0.88	4.76	0.64
8	1.00	1.09	5.47	0.69	1.03	1.08	5.13	0.73	1.11	0.94	5.17	0.63
9	1.21	1.25	5.94	0.73	1.23	1.26	5.93	0.74	1.34	1.02	6.02	0.58
High(H)	1.83	1.69	7.29	0.80	1.89	1.75	7.51	0.81	1.99	1.46	7.40	0.68
H-L	1.86	2.12	6.13	1.20	2.01	2.26	5.80	1.35	2.22	1.97	5.93	1.15

In this table, we report the performance of prediction-sorted portfolios over the 30-year out-of-sample testing period. All stocks are sorted into deciles based on their predicted returns for the next month. Columns “Pred,” “Avg,” “SD,” and “SR” provide the predicted monthly returns for each decile, the average realized monthly returns, their standard deviations, and Sharpe ratios, respectively. All portfolios are value weighted.

Results

- “for each method, we calculate the reduction in R2 from setting all values of a given predictor to zero within each training sample and average these into a single importance measure for each predictor.
- “all methods agree on a relatively small set of dominant predictive signals, [...] associated with price trends including return reversal and momentum [...], stock liquidity, stock volatility, and valuation ratios.

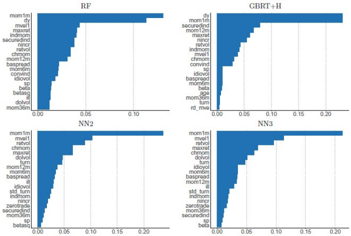


Figure 4
Variable importance by model
Variable importance for the top-20 most influential variables in each model. Variable importance is an average over all training samples. Variable importance within each model is normalized to sum to one.

Case studies

Option Pricing

- Recall: The value of a call option for a non-dividend-paying underlying stock in terms of the Black–Scholes parameters is:

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

- V is the price of the option as a function of stock price S and time t, r is the risk-free interest rate, and σ is the volatility of the stock. $N(\cdot)$ is the standard normal cumulative distribution function.
- Can machine learning methods **learn** the Black-Scholes equation after observing different specifications and corresponding prices?

Start with simulated data

- We compute option prices for Call options for a grid of different combinations of maturity (`T`), risk-free rate (`r`), volatility (`sigma`), the strike price (`K`), and current stock price (`S`)
- To make it harder: Add an idiosyncratic error term to each observation

“Learning” Black-Scholes

- Implementation left to you
- Considered models: Deep Neural Network, Lasso (based on a polynomial expansion of all inputs), Random Forests, and a single-layer neural network
- Below: Out-of-sample prediction error for different strike prices

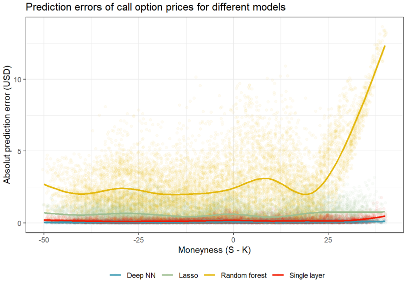
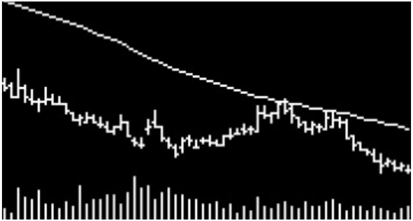


Image recognition

- Jiang et al. (2022) “train” a neural network on financial time series charts
- “reconsider the idea of trend-based predictability using methods that flexibly learn price patterns that are most predictive of future returns, rather than testing hypothesized or pre-specified patterns (e.g., momentum and reversal).”



- Charts can be represented as vectors with length *horizon* × *resolution*
- Convolutional neural network: cross-parameter restrictions that dramatically reduce parameterization relative to standard feed-forward neural networks
- Replication code available on tidy-finance.org