Machine Learning in Asset Management—*Part 1: Portfolio Construction—Trading Strategies*

DEREK SNOW

DEREK SNOW
is a doctoral candidate of finance at the University of Auckland in Auckland, New Zealand.
d.snow@firmai.org

KEY FINDINGS

- Machine learning can help with most portfolio construction tasks like idea generation, alpha factor design, asset allocation, weight optimization, position sizing, and the testing of strategies.
- This is the first in a series of articles dealing with machine learning in asset management and more narrowly on trading strategies equipped with machine-learning technologies.
- Each trading strategy can end up using multiple machine learning frameworks. The author highlights nine different trading varieties each making use of a reinforcement-, supervised-, or unsupervised-learning framework or a combination of these learning frameworks.

ABSTRACT: *This is the first in a series of articles dealing with machine learning in asset management. Asset management can be broken into the following tasks: (1) portfolio construction, (2) risk management, (3) capital management, (4) infrastructure and deployment, and (5) sales and marketing. This article focuses on portfolio construction using machine learning. Historically, algorithmic trading could be more narrowly defined as the automation of sell-side trade execution, but since the introduction of more advanced algorithms, the definition has grown to include idea generation, alpha factor design, asset allocation, position sizing, and the testing of strategies. Machine learning, from the vantage of a decision-making tool, can help in all these areas.*

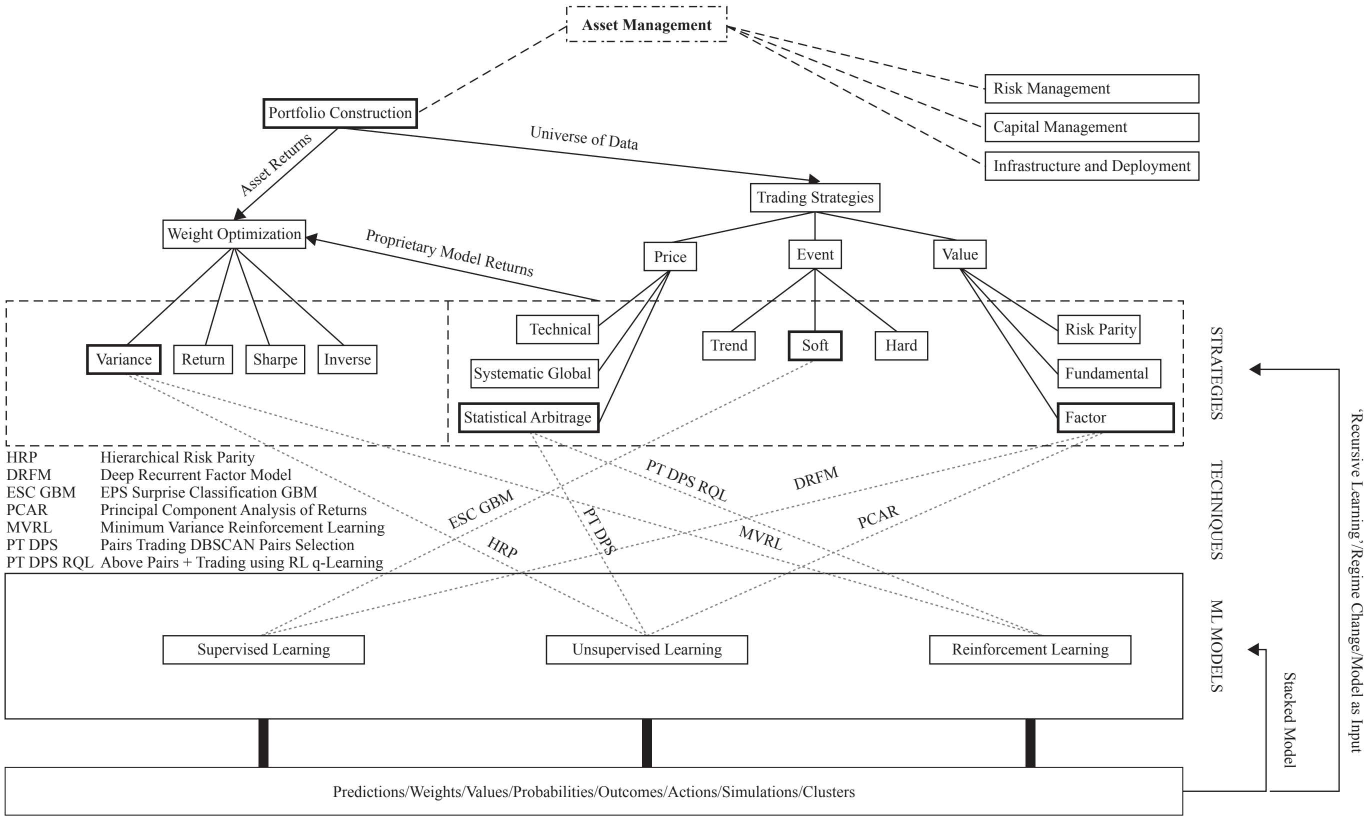
TOPICS: ***Big data/machine learning, analysis of individual factors/risk premia, portfolio construction, performance measurement****

Financial machine learning research can loosely be divided into four streams. The first concerns asset price prediction where researchers attempt to predict the future value of securities using a machine learning methodology. The second stream involves the prediction of hard or soft financial events like earnings surprises, regime changes, corporate defaults, and mergers and acquisitions. The third stream entails the prediction and/or estimation of values that are not directly related to the price of a security, such as future revenue, volatility, firm valuation, credit ratings, and factor quantiles. The fourth and last stream comprises the use of machine learning techniques to solve traditional optimization and simulation problems in finance like optimal execution, position sizing, and portfolio optimization.

The first three streams are concerned with the creation of trading strategies,

Exhibit 1

Financial Machine Learning in Portfolio Construction



HRP	Hierarchical Risk Parity
DRFM	Deep Recurrent Factor Model
ESC GBM	EPS Surprise Classification GBM
PCAR	Principal Component Analysis of Returns
MVRL	Minimum Variance Reinforcement Learning
PT DPS	Pairs Trading DBSCAN Pairs Selection
PT DPS RQL	Above Pairs + Trading using RL q-Learning

and the last stream is concerned with everything else, like weight optimization, optimal execution, risk management, and capital management. Within these streams we can make use various machine learning techniques. These techniques can be broken into the following: (1) processing of unstructured data, (2) supervised learning, (3) validation techniques, (4) unsupervised learning, and (5) reinforcement learning. Every machine learning solution is constructed out of a mixture of these. In this series, we look at the application of these techniques in asset management.

Exhibit 1 outlines a few different ways in which machine learning can be used in portfolio construction. Portfolio construction can broadly be broken into trading strategies[1] and weight optimization. In the first part of this series we look at trading strategies and in the second part we look at weight optimization. The trading strategy styles in the first three streams of financial machine learning research, *price*, *event*, and *value*, can be split into unique trading themes depending on the data used and the outcome one is trying to predict. *Price* strategies include technical, systematic global macro, and statistical arbitrage, because of the central role price has to play in the input data and predicted outcomes. *Event* strategies include trend, soft-event, and hard-event themes, because of the need to predict a change. *Value* strategies include risk parity, factor investing, and fundamental themes, because these measures estimate intermediary values not directly related to the asset price.

Each trading theme can end up using different machine learning frameworks. For example, technical and statistical arbitrage strategies can use a supervised- or reinforcement-learning approach or a combination of both, and factor investing strategies can use a supervised- or unsupervised-learning approach. The best labeling

[1] This article's primary focus is on showing how machine learning can be used to develop trading strategies as opposed to the performance of the trading strategies.

practice or naming convention for machine learning trading strategies would be a combination of the trading theme, the method, and the submethod used. The submethod drives one level deeper than the machine learning framework; for a reinforcement-learning framework, the submethod would for example be policy optimization, Q-learning, or model-based approaches. In this article, I will not provide information on data-processing techniques like natural language processing, image and voice processing, and feature generation; however, it should be noted that these techniques can form part of reinforcement-, supervised-, and unsupervised-learning frameworks.

It is necessary to define the difference between the aforementioned themes. *Technical trading* is the use of market data and its transformations to predict the future price of an asset. *Trend trading* involves strategies where one takes a position in the asset only after predicting a change in trend. *Statistical arbitrage* seeks mispricing by detecting asset relationships and/or potential anomalies, believing the anomaly will return to normal. *Risk parity* strategies diversifies across assets according to the volatility they exhibit; when one asset class's volatility exceeds another rebalancing can occur by selecting individual units within each asset class or simply by using leverage. *Event trading* involves the prediction of *hard* or *soft* financial events like corporate defaults, mergers and acquisitions, and earnings surprises. *Factor investing* attempts to buy assets that exhibit a trait historically associated with promising investment returns. *Systematic global macro* relies on macroeconomic principles to trade across asset classes and countries. *Fundamental trading* relies on the use of accounting, management, and sentiment data to predict whether a stock is over or undervalued. In this section, I will provide machine learning applications for some of these trading themes.

As soon as you have trained your machine learning model, you can decide whether you want to use the model as part of a greater ensemble of models to create an improved final prediction model that would be used for trading purposes. This ensemble of models can additionally pass through a second supervised machine learning model that would decide on the most profitable model weighting scheme; this is known as a stacked model. Once you have devised a few independently stacked trading models, you can pass the proprietary model returns to an unsupervised-learning portfolio-weight–optimization scheme like hierarchical risk parity for the final strategy allocation. It is currently feasible to substitute a large portion of traditional algorithmic trading techniques with their machine learning equivalents.

It is possible to construct a strategy that learns all the way down. For example, although one can create one reinforcement-learning agent that is able to ingest a lot of data and make profitable decisions inside an environment, you are often better off to create more simple agents at the lower level while pyramiding additional decision-making responsibilities upward. Lower-level agents can look at pricing and fundamental and limited capital market data, whereas a meta-agent can select or combine strategies based on potential regime shifts that happen at the economic level and only make the trading decisions then.[2]

A meta-learner can choose between a few hundred models based on the current macro-regime. At the end, all the meta learners, or depending on how deep you go, meta-cubed learners, should form part of the overall portfolio. The core function is to carry our portfolio-level statistical arbitrage to the outmost extreme using all the tools available financial or otherwise. You need not use an additional level of reinforcement- or supervised-learning algorithms; you can also use unsupervised clustering algorithms. You can discover multiple economic regimes by using k-nearest neighbors (KNN) clustering, an unsupervised-learning technique, to select the potential regime of the last 30 days. Then one can select strategy by looking at the historic success across all regime types.[3]

Machine learning is limitless in the sense that you can tweak it endlessly to achieve some converging performance ceiling. Some of these tweaks include the different methods to perform validation, hyperparameter selection, up-and down sampling, outlier removal, data replacement, and so on. Features can also be transformed in myriad ways; the dimensions of features can be reduced or inflated; variables can be generated through numerous unsupervised methods; variables can

[2] As a result, it is not always necessary, or optimal, to include all the data at the lower modeling level, and it therefore becomes an optimization question in itself to decide at what modeling level to use the data in your hierarchical modeling structure.

[3] While doing this, pay attention to the stability of clusters. The assignments might not persist in time series; mini-batch and ward clustering algorithms tend to be more persistent.

also be combined, added, or removed; models can be fed into models; and on top of that it is possible to use all machine-learning frameworks (i.e., supervised, reinforcement, and unsupervised learning) within a single prediction problem. The only way to know whether any of these adjustments are beneficial is to test it empirically on validation data.

How do we know if any of these adjustments would lead to a better model? Most of the time we can use proxies for potential performance like the Akaike information criterion (AIC) or feature–target correlation. These approaches get us halfway toward a good outcome. The best approach is to retest the model each time a new adjustment is introduced. The tests should not be performed on the data that would be used in testing the performance of the model (i.e., the holdout set); instead a separate validation set should be specified for this purpose. It is also preferable to change the validation data after each new empirical test to ensure that these adjustments do not overfit the validation set; one such approach is known as K-fold cross-validation, where the validation set is randomly partitioned into K equal-sized subsamples for each test.

In this article, we survey a few ways in which machine learning can be used to enhance or even create trading strategies. As stated, we are only limited by our imagination when devising machine learning strategies. In this section, we highlight nine different trading varieties that make use of a reinforcement-, supervised-, or unsupervised-learning framework or a combination of learning frameworks. I named these strategies Tiny RL, Tiny VIX CMF, Agent Strategy, Industry Factor, Global Oil, Earnings Surprise Prediction, Deep Trading, Stacked Trading, and Pairs Trading. I organized them according to machine-learning framework, and each strategy is titled by name, theme, method, and submethod.

REINFORCEMENT LEARNING

Reinforcement learning (RL) in finance comprises the use of an agent that learns how to take actions in an environment to maximize some notion of cumulative reward. We have an agent that exists in a predefined environment; the agent receives as input the current state S_t and is asked to take an action A_t to receive a reward R_{t+1}, the information of which can be used to identify the next optimal action, A_{t+1}, given the new state S_{t+1}. The final objective function can be the realized/unrealized profit and loss and even a risk-adjusted performance measure like the Sharpe ratio.

Tiny RL—Technical/RL/Policy

In this example we will make use of gradient descent to maximize a reward function.[4] The Sharpe ratio will be used as the reward function. The Sharpe ratio is used as an indicator to measure the risk-adjusted performance of an investment over time. Assuming a risk-free rate of zero, the Sharpe ratio can be written as

$$S_T = \frac{A}{\sqrt{B - A^2}} \tag{1}$$

Further, to know what percentage of the portfolio should buy the asset in a long-only strategy, we can specify the following function, which will generate a value between 0 and 1:

$$F_t = tanh(\theta^T x_t) \tag{2}$$

The input vector is $x_t = [1, r_{t-M}, \ldots, F_{t-1}]$ where r_t is the percent change between the asset at time t and $t - 1$ and M is the number of time-series inputs. This means that at every step the model will be fed its last position and a series of historical price changes that are used to calculate the next position. Once we have a position at each time step, we can calculate our returns R at each time step using the following formula. In this example, δ is the transaction cost:

$$R_t = F_{t-1} r_t - \delta \left| F_t - F_{t-1} \right| \tag{3}$$

To perform gradient descent, one must compute the derivative of the Sharpe ratio with respect to theta, or $\frac{dS_T}{d\theta}$ using the chain rule and the previous formula. It can be written as

$$\frac{dS_T}{d\theta} = \sum_{t=1}^{T} \left(\frac{dS_T}{dA}\frac{dA}{dR_t} + \frac{dS_T}{dB}\frac{dB}{d\theta} \right) \cdot \left(\frac{dR_t}{dF_t}\frac{dF}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta} \right) \tag{4}$$

Please see the online supplement for the code.

[4] Thank you to Teddy Koker for developing this easy-to-follow method.

Tiny VIX CMF—StatArb/RL/Policy

CBOE Volatility Index (VIX) and Futures on the Euro STOXX 50 Volatility Index (VSTOXX) are liquid and so are exchange-traded notes/exchange-traded funds (ETNs/ETFs) on VIX and VSTOXX.[5] Prior research has shown that the future curves exhibit stationary behavior with mean reversion toward a contango. First, one can imitate the futures curves and ETN price histories by building a model to manage the negative roll yield. The constant-maturity futures (CMF) can be specified as follows:

Denote $\theta = T - t$ to have constant maturity, $V_t^\theta = F_{t,t+\theta}$, for $t \leq T_1 \leq t + \theta \leq T_2$ define

$$a(t) = \frac{T_2 - (t-\theta)}{T_2 - T_1} \tag{5}$$

Note that:

- $0 \leq a(t) \leq 1$
- $a(T_1 - \theta) = 1$ *and* $a(T_2 - \theta) = 0$
- *linear in t*

The CMF is the interpolation,

$$V_t^\theta = a(t)F_t^{T1} + (1 - a(t))F_t^{T2} \tag{6}$$

where V_t^θ is a stationary time series.

One can then go on to define the value of the ETN so that you take the roll yield into account. I wanted to focus on maturity and instrument selection and therefore ignored the roll yield and simply focused on the CMFs. But, if you are interested, the value of the ETN can be obtained as follows:

$$\frac{dl_t}{l_t} = \frac{a(t)dF_t^{T1} + (1-a(t))dF_t^{T2}}{a(t)F_t^{T1} + (1-a(t))F_t^{T2}} + rdt \tag{7}$$

where r is the interest rate.

Unlike the Tiny VIX CMF approach, this strategy makes use of numerical analyses before a reinforcement learning step. First, out of all seven securities (J), establish a matrix of 1 and 0 combinations for simulation purpose to obtain a matrix of $2^7 - 2 = 126$ combinations. Then use a standard normal distribution to randomly assign weights to each value in the matrix. Create an inverse matrix and do the same. Now normalize the matrix so that each row equals 1 to force neutral portfolios. The next part of the strategy is to run this random-weight assignment simulation N (600) number of times, depending on your memory capacity, because this whole trading strategy is serialized. Thus, each iteration (N) produces normally distributed long and short weights (W) that have been calibrated to initial position neutrality (long weights = short weights); the final result is 15,600 trading strategies.

The next part of this system is to filter out strategies with the following criteria. Select the top X% of strategies for their highest median cumulative sum over the period. From that selection, select the top Y% for the lowest standard deviation. Of that group, select Z% again for the highest median cumulative sum strategies. X, Y, and Z are risk–return parameters that can be adjusted to suit your investment preferences. In this example, they are set at 5%, 40%, and 25% respectively. It is possible to efficiently select these parameters by adding them to the reinforcement learning action space. Of the remaining strategies, iteratively remove highly correlated strategies until only 10 (S) strategies remain. With those remaining 10 strategies, which have all been selected using only training data, use the training data again to formulize a reinforcement learning strategy using a simple multilayer perceptron (MLP) neural network with two hidden layers to select the best strategy for the specific month by looking at the last 6 months returns of all the strategies (i.e., 60 features in total). Finally test the results on an out of sample test set. Note in this strategy no hyperparameters selection was done on a development set, as a result, it is expected that results can further be improved.

Please see the online supplement for the code.

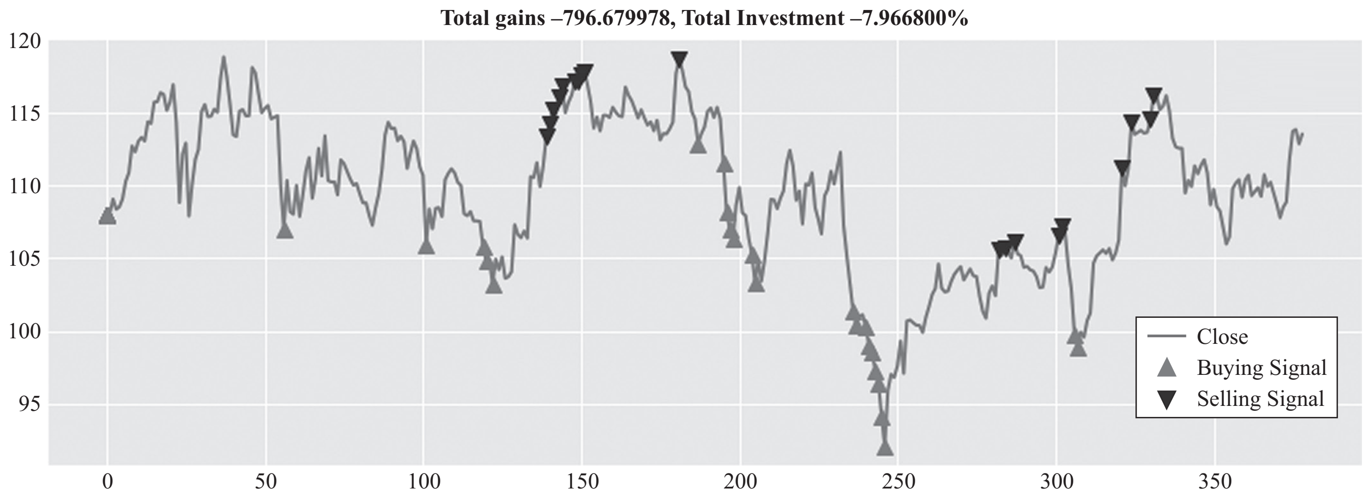
Agent Strategy—Price/RL/ Various Submethods

Here, more than 20 reinforcement-learning submethods are developed using different algorithms. The first three in the code supplement do not make use of RL; their rules are determined by arbitrary inputs. This includes a turtle-trading agent, a moving-average agent, and a signal-rolling agent. The rest of the coding notebook contains progressively more

[5] Thank you to Andrew Papanicolaou at NYU for his initial assistance to develop this method.

EXHIBIT 2

Example of a Reinforcement Learning Strategy's Performance



involved reinforcement-learning agents. The notebook investigates, among others, policy-gradient agents, Q-learning agents, actor–critic agents, and some neuro-evolution agents and their variants. With enough time, all these agents can be initialized, trained, and measured for performance. Each agent individually generates a chart that contains some of the performance information, as shown in Exhibit 2.

In this section we will look at three of the most popular methods, Q-learning, policy gradient, and actor–critic. Some quick mathematical notes: s = states, a = actions, r = rewards. In addition, action-value functions Q, state-value functions V, and advantage functions A are defined as

$$Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t] \tag{8}$$

$$V^{\pi}(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)}[Q^{\pi}(s_t, a_t)] \tag{9}$$

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \tag{10}$$

Then, $\widehat{V_{\varnothing}^{\pi}}(s_t)$ is the fitted value function for $V^{\pi}(S_t)$.

Q-Learning is an online action-value function learning with an exploration policy, such as epsilon–greedy.[6] You take an action, observe, maximize, adjust policy and do it all again.

Take some action a_i and observe (s_i, a_i, s_i', r_i):

$$y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\theta(s_i', a_i') \tag{11}$$

$$\varnothing \leftarrow \varnothing - \alpha \frac{dQ_\varnothing}{d\varnothing}(s_i, a_i)(Q_\theta(s_i, a_i) - y_i)$$

Then explore with the epsilon–greedy policy:

$$\pi(a_t|a_t) = \begin{cases} 1 - \epsilon & \mathit{if} \quad a_t = \mathit{argmax}_{a_t} Q_\theta(s_t, a_t) \\ & \epsilon/(|A| - 1) \ \mathit{otherwise} \end{cases} \tag{12}$$

With policy gradients, you maximize the rewards by taking actions where higher rewards are more likely.

$$\text{Sample } \{\tau^I\} \text{ from } \pi_\theta(a_t|s_t)$$
$$\nabla_\theta J(\theta) \approx \sum_i \left(\sum_i \nabla_\theta \log\pi_\theta(a_t^i|s_t^i)\right)\left(\sum_i r(s_t^i, a_t^i)\right)$$
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{13}$$

[6] Epsilon–greedy is very simple: Keep track of the average payout of each strategy and select the strategy with the highest current average payout. It is therefore an exploitive as opposed to an explorative policy.

Actor–critic is a combination of policy-gradient and value-function learning. In this example, I focus on the online as opposed to the batch model.

$$\begin{gathered}\text{Take action} \sim \pi_\theta(a|s),\ \text{get}(s,a,s',r)\\ \text{Update } \widehat{V_\varnothing^\pi} \text{ using target } r+\gamma\widehat{V_\varnothing^\pi}(s')\\ \text{Evaluate } \widehat{A^\pi}(s,a)=r(s,a)+\gamma\widehat{V_\varnothing^\pi}(s')-\widehat{V_\varnothing^\pi(s)}\\ \nabla_\theta J(\theta)\approx\nabla_\theta log\pi_\theta(a|s)\widehat{A^\pi}(s,a)\\ \theta\leftarrow\theta+\alpha\nabla_\theta J(\theta)\end{gathered} \tag{14}$$

Please see the online supplement for the code.

SUPERVISED LEARNING

Supervised-learning (SL) techniques are used to learn the relationship between independent attributes and a designated dependent attribute. *SL* refers to the mathematical structure describing how to make a prediction $\boldsymbol{y}_i$ given $\boldsymbol{x}_i$. Instead of learning from the environment like RL, SL methods learn the relationships in data. All supervised learning tasks are divided into classification or regression tasks. Classification models are used to predict discrete responses (e.g., binary 1, 0; multiclass 1, 2, 3). Regression is used for predicting continuous responses (e.g., 3.5%, 35 times, $35,000). In the examples that follow, we will use both classification and regression models.

Industry Factor—Factor/SL/LASSO

In this example, we will look at the use of machine learning tools to analyze industry return predictability based on lagged industry returns across the economy (Rapach et al. 2019). A strategy that longs the highest and shorts the lowest predicted returns returns an alpha of 8%. In this approach, one has to be careful about multiple testing and postselection bias. A LASSO (Least Absolute Shrinkage and Selection Operator) regression[7] is eventually used in a machine-learning format to weight industry importance, but before that we should first formulate a standard predictive regression framework:

$$\boldsymbol{y_i}=a_i^*\boldsymbol{\gamma_T}+\boldsymbol{Xb_i^*}+\boldsymbol{\varepsilon_i}\quad for\ i=1,\ldots,N, \tag{15}$$

[7]LASSO regressions shrink regression coefficients toward zero to encourage simple sparse models.

where

$$\begin{gathered}y=[r_{i,1}\ldots r_{i,T}];\ X=[x_1\ldots x_N];\\ x_j=[r_{i,0}\ldots r_{i,T-1}]\ for\ j=1,\ldots,N\\ b_i^*=[b_{i,1}^*\ldots b_{i,N}^*];\ \varepsilon_i=[\varepsilon_{i,1}\ldots\varepsilon_{i,T}]\end{gathered} \tag{16}$$

In addition, the LASSO objective ($\boldsymbol{\gamma_T}$) can be expressed as follows, where ϑ_i is the regularisation parameter.

$$\underset{a_1\in\boldsymbol{R},\ b_i\in R^N}{\arg min}\left(\frac{1}{2T}\left\|\mathrm{y}_i-a_i\boldsymbol{\gamma_T}-\boldsymbol{Xb_i}\right\|_2^2+\vartheta_i\left\|\boldsymbol{b_i}\right\|_1\right) \tag{17}$$

The LASSO regression generally performs well in selecting the most relevant predictor variables. Some argue that the LASSO penalty term overshrinks the coefficient for the selected predictors. In that scenario, one can use the selected predictors and re-estimate the coefficients using ordinary least squares (OLS). This submodel—an OLS regression model in this case—can be replaced by any other machine-learning regressor. In fact, the main model and submodel can both be machine-learning regressors, the first selecting the features and second predicting the response variable based on those features.

Please see the online supplement for the code.

Global Oil—Systematic Macro/SL/Elastic Net

When oil exits a bear market, the currency of oil-producing nations should also rebound. With this strategy, we will investigate the effect the price of oil has on the Norwegian krone (NOK) and identify whether a profitable trading strategy can be executed. To start we need a stabilizer currency to regress against. The currency should be unrelated to the currency under investigation. Something like the Japanese yen (JPY) is a good candidate. From here on, one would use the price of the NOK and Brent as measured against JPY to identify whether the Norwegian currency is under- or overvalued. We will use an elastic net regression as the machine-learning technique. It is a good tool when multicollinearity is an issue.

An elastic net is a regularized[8] regression method that combines both L1 (LASSO) and L2 (Ridge) penalties.[9] The estimates from the elastic net method are defined by

$$\hat{\beta} = \underbrace{argmin}_{\beta}\left(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|B\|_1\right) \tag{18}$$

The loss function becomes strongly convex as a result of the quadratic penalty term, therefore providing a unique minimum. Now that the predictors are in place, one has to set up a pricing signal; one sigma two-sided is the common practice in arbitrage. We short if it spikes above the upper threshold and long on the lower threshold. The stop-loss will be set at two standard deviations. At that point, one can expect the interpretation of the underlying model to be wrong and therefore choose to exit the position.

Please see the online supplement for the code.

Earnings Surprise Prediction—Soft-Event/SL/Gradient Boosting Machines

Here we investigate an earnings prediction strategy. It is a classification task where the response variable for the machine-learning model is the occurrence of an earnings surprise. An earnings surprise is simply defined as a percentage change from the analyst's earnings per share (EPS) expectation and the actual EPS that crosses a predefined threshold *s*. The percentage threshold, *s*, expresses the magnitude of the surprise.

$$X = \frac{EPSAC_{it} - EPSAN_{it}}{EPSAN_{it}} - 1 \tag{19}$$

$$SURP_{itsx} = 0,\ where\ X < -S, \text{Negative} \tag{20}$$

$$SURP_{itsx} = 1,\ where - S \le X \le S,\ \text{Neutral} \tag{21}$$

$$SURP_{itsx} = 2,\ where\ X > S, \text{Positive} \tag{22}$$

[8] *Regularization* is any method that decreases the complexity of the model with the hope of improving its out-of-sample performance.

[9] The biggest difference between L1 and L2 is that L1's penalty is equal to the absolute value of the coefficient and L2's penalty is equal to the square of the magnitude; as a result, L1 can eliminate coefficients and L2 can only shrink coefficients without eliminating them.

To provide some clarity, *i* is the *i*th firm in the sample, *t* is the time of the quarterly earnings announcement, *s* is the respective constant surprise threshold, *x* is a constant percentage of the sample of earnings announcements sorted by date, *EPSAN* is analyst earnings per share consensus forecast, and *EPSAC* is the actual earnings per share as reported by the firm.

Following is high-level pseudo-code to provide a better understanding of some of the core concepts of the developed black-box model and its relationship with the training set, test set, prediction values, and metrics:

```
1. $Classifier = XGBoostTreeClassifier\ (Train_X,\ SURP_{its(0:x)},\ Valid,\ Param)$
2. $PredSURP_{ith} = Classifier(Test_X)$
3. $Metrics = Functions(SURP_{its(x:1)},\ PredSURP_{ith})$
```

Using the predictions $PredSURP_{ith}$, you can expresses a simple strategy by going long (short) on stocks that are expected to experience a positive (negative) surprise tomorrow (*t*), at closing today ($t - 1$), and liquidating the stocks at closing tomorrow (*t*). The stocks are equally weighted to maintain well-diversified returns for the day because there are, on average, only four firms in a portfolio of expected surprises, but there can be as few as one firm in a portfolio. For each day, we form stocks into positive and negative surprise prediction portfolios for surprises that deviate from −50% to 50% to select the best performing threshold. The preferred threshold is selected based on tests done against a validation set. The results in the validation set show that the best trading strategies exist between 5%–20%, with 15% being the optimal trading strategy for positive surprises.

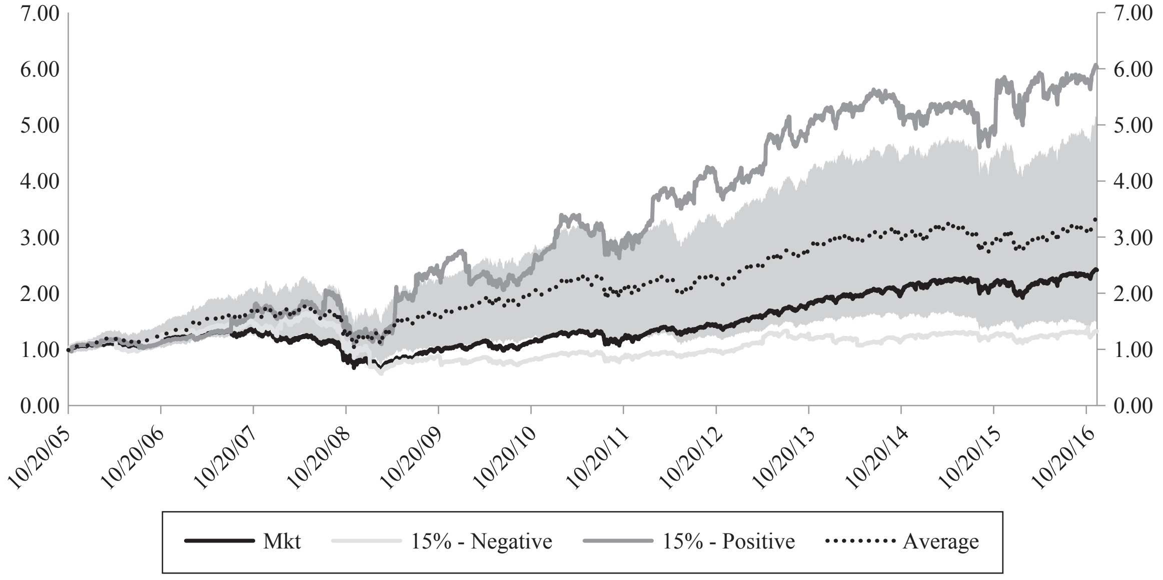
The strategies recommended in this section fully invest all capital in each event. It is therefore important to include some sort of loss minimization strategy. As a result, one strategy incorporates a stop-loss for stocks that fell more than 10%. Here 10% is only the trigger, and a conservative loss of 20% is used to simulate the slippage:

$$R_{it} = \frac{(S_{it} - S_{i,(t-1)})}{S_{i,(t-1)}},\ if\ \ \frac{Sl_{it} - S_{i,(t-1)}}{S_{i,(t-1)}} < -10\%,\ R_{it} = -20\% \tag{23}$$

The equal weighted return of a portfolio of surprise firms is then calculated as

$$R_{pt} = \frac{1}{n}\sum_{i=1}^{n_{pt}} R_{it},\ where\ n = 0,\ R_{pt} = R_{Mt}, \tag{24}$$

EXHIBIT 3
Portfolio Value 15% Surprise Prediction Strategy



In this equation, i is all the firms that experience surprises on date t. Therefore, R_{it} is the return on the common stock of firm i on date t and n_{pt} is the number of firms in portfolio p at the close of the trading on date $t-1$. R_{Mt} is the market return rate.

Exhibit 3 reports the cumulative portfolio returns of buying and holding positive and negative surprises predictions for all firms with a market value of $10 billion or more. On average, there are about four firms for each portfolio day. On days where no trading surprises occur, a position in the market is taken. The band in the middle is the 99% significance band obtained from 1,000 Monte Carlo simulations that randomly take a position in firms before an earnings announcement. In total there are 2,944 trading days for the long strategy; 215 of these days are returns from earnings surprises comprising 774 firms, and the rest are simple market return days.

Deep Trading—Technical/SL/Various DL

There are 30 different neural network submethods investigated here.[10] This includes Vanilla RNN, GRU, LSTM, Attention, DNC, Byte-net, Fairseq, and CNN methods.[11] The mathematics of the different frameworks are vast and would take too much space to include here. None of these methods have been turned into trading strategies yet. Here, we are simply predicting the future price of the stock, so the models can easily be transformed into directional trading strategies from this point. You can construct the trading policies by hand or rely on reinforcement learning strategies to develop the best trading policies.

Exhibit 4 can help us to understand the major differences between the submethods. A Vanilla RNN uses the simple multiplication of inputs ($\boldsymbol{x}_t$) and previous outputs (h_{t-1}) passed through a tanh activation function. A GRU introduces the additional concept of a gate that decides whether to pass a previous output (h_{t-1}) to a next cell in an attempt to solve the vanishing gradient problem.[12] It is simply an additional mathematical operation performed on the same inputs. With the LSTM an additional gate is introduced to the GRU method.

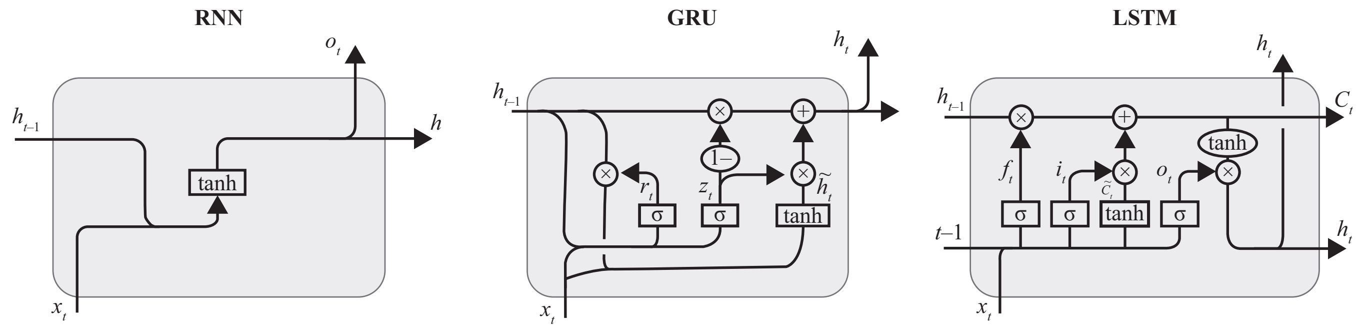
[10] None of the supervised learning strategies have been turned into trading strategies yet. Here we are simply predicting the price of the stock a few days in advance, so the models can easily be transformed into directional trading strategies from this point.

[11] Listed in order: RNN = recurrent neural network, GRU = gated recurrent unit, LSTM = long short-term memory, Attention = attention mechanism, DNC = differentiable neural computer, CNN = convolutional neural network.

[12] A difficulty found in artificial neural networks that may completely stop the neural network from training further.

EXHIBIT 4

Architecture of RNN, GRU, and LTSM Cells



Again, these are additional mathematical operations on the same inputs. Moving from RNN to LSTM we are simply introducing more control knobs for the flow and mixing of input data to establish the final weights. The LSTM method is designed to focus on establishing weights that maintain information that persist for longer periods. The codes of these three methods and many others are available in the online supplement.

Please see the online supplement for the code.

Stacked Trading—Technical/SL/Stacked

Stacked trading is purely experimental; it involves the training of multiple models (base learners or level 1 models), after which they are weighted using an extreme gradient-boosting model (metamodel or level 2 model). In the first stacked model, which I will refer to as EXGBEF, we use autoencoders to create additional features. In the second model, DFNNARX, autoencoders are used to reduce the dimensions of existing features. In the second model, I include additional economic (130+ time series) and fund variables to the stock price variables. Similar to the deep trading example, we have price movement predictions, but we have not developed a trading policy yet. Exhibit 5 graphically shows the concept of stacking.

The training data X has m observations and n features. There are M different models that are trained on X. Each model provides predictions $\hat{y}$ for the outcome y, which are then cast into second-level training data $X^{(l2)}$, which is now $m \times M$ sized. The M predictions become features for this second-level data. A second-level model (or models) can then be trained on these data to produce the final outcomes $\hat{y}^{fin}$, which will be used for predictions. With stacking it can help to use out-of-sample training data at each modeling level; otherwise the nth level model will be biased to use only the best-performing model in the previous modeling level.

Please see the online supplement for the code.

UNSUPERVISED LEARNING

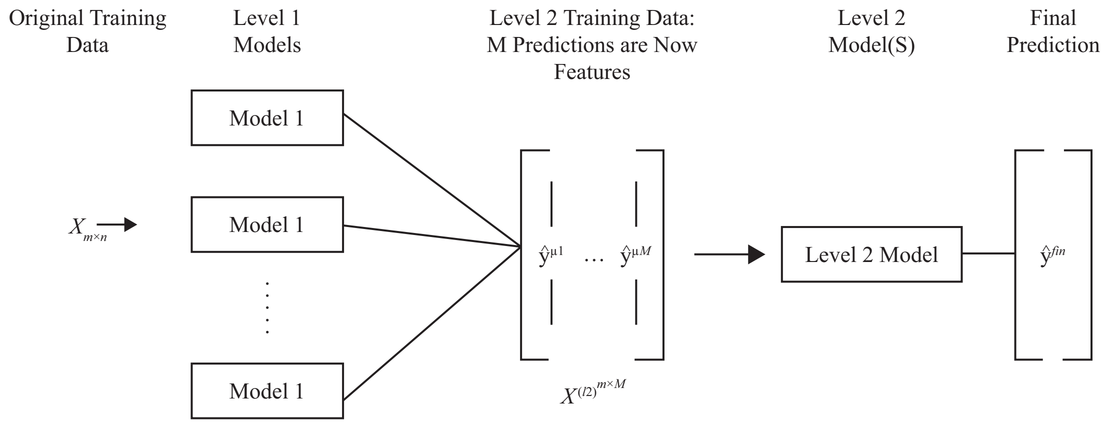
Unlike SL, which finds patterns using both input data and output data, unsupervised learning (UL) methods finds patterns using only input data. An unsupervised-learning framework is useful when you are not quite sure what to look for. It is often used for the exploratory analysis of raw data and for problem discovery purposes. Most UL techniques take the form of dimensionality reduction or cluster analysis where you group data items that have some measure of similarity based on characteristic values. Two of the most important techniques are K-means clustering and principal component analysis (PCA).

PCA attempts to reduce the number of *features* while preserving the variance, whereas clustering reduces the number of *data points* by summarizing them according to their mean expectations. However, those clusters assignments can also be used to label each data point with its assigned cluster, leading to a dimensionality reduction toward only one feature. K-means and PCA in some sense maximize a similar objective function with K-means having an additional categorical constraint. The only requirement to be called an unsupervised learning strategy is to learn a new feature space that captures the characteristics of the original space by maximizing some objective function.[13]

[13] Inherent to the PCA is the maximization of variance through a simple linear algebra operation by taking the eigenvectors of a covariance matrix of features.

Exhibit 5

Architecture of Stacked Models



Pairs Trading—StatArb/UL/K-Means

Pairs trading is a simple statistical arbitrage strategy that finds pairs that exhibit similar historic price behavior and then, once they diverge, betting on the expectation that they will converge. With unsupervised-learning methods, we can add an extra safeguard by selecting stocks that are structurally similar. In a universe of assets over time, X_t, identify K_t features that help to define the characteristic of the company. One can use features such as the price-to-earnings ratio, dividend yield, and return on assets, as well as some market-based data like lagged returns and technical indicators.

You can then use the data to perform K-mean clustering to reduce the distance between data points around a predefined number of clusters. Within each cluster pick the respective assets X_t^x, so that $corr(X_t^{x_1}, X_t^{x_2})$ exceeds some threshold P. We can let X_t^1 and X_t^2 denote the prices of two correlated stocks, if we believe in the mean-reverting nature and the ground truth of the shared correlation, we can initiate a mean-reverting process (Z_t) as follow, $Z_t = X_t^1 - K_0 X_t^2$, where K_0, is a positive scalar such that at initiation $Z_t = 0$.

This mean reverting process, i.e., the change in mean, is governed by $dZ_t = a(b - Z_t)dt + \sigma dW_t, Z_0 = 0$, where $a > 0$ is the rate of reversion, b the equilibrium level, $\sigma > 0$ the volatility, and W_t the standard Brownian motion. This Brownian motion exists within the mean-reverting process; hence it cannot have drift. Z_t is thus the pairs position: 1 share long in X_t^1 and K_0 shares short in X_t^2; and dZ_t describes the underlying dynamic under strict mathematical assumptions.

Given the construction, we expect the prices of assets to revert to normal. For all the clustered pairs, we can specify an easy and arbitrary trading rule, which specifies that if $abs(X_t^1 - K_0X_t^2)/(\frac{1}{n}\Sigma_{t=1}^{n} abs[X_t^1 - K_0X_t^2]) > 0.1$, then you sell the high-priced asset and buy the low-priced asset, and you do the reverse as soon as the formula hits 0.5, whereby you assume that this might be the new natural relational equilibrium. In practice we also impose additional state constraints—for example, we require, $Z_t \geq M$, where M is the stop-loss level in place for unforeseeable events and to satisfy a margin call; further alterations should include transaction and borrowing costs.

Pair selection and trading logic can be much more involved than what we just described. Instead of using simple trading rules, dynamic model-based approaches can be used to define the trading logic. The previously mentioned mean reversion process can be discretized into an AR process where the parameters can be estimated iteratively and be used in the trading rule. This is called the stochastic spread method.

In addition, instead of performing the final pair selection using distance, rolling OLS cointegration or Kalman filter cointegration, unsupervised-learning methods like hierarchical clustering, K-means, variational autoencoder embedding, or DBSCAN[14] can be

[14]DBSCAN = density-based spatial clustering of applications with noise.

used. Although an unsupervised-learning method might not be the best method to directly select pairs, it is a good intermediate step to create clusters with K-means out of which traditional pairs selection strategies can be used.

SUPERVISED LEARNING VERSUS REINFORCEMENT LEARNING

The general pipeline for supervised machine-learning trading involves the acquisition of data, processing of data, prediction, policy development, backtesting, parameter optimization, live paper simulation, and finally live trading of the strategy. The basic supervised learning task involves some form of price prediction. This includes regressors that predict the price level and classifiers that predict price direction and magnitude in predefined classifications for future time steps. Supervised machine-learning models, especially neural networks, can keep up with changing market regimes as long as they are able to do online training.[15] The reason supervised-learning processes tend to fail is because the iterative steps from ML prediction through to policy development, backtesting, and parameter optimization are fragile, slow, and prone to error. A further issue is that the performance simulation turns up too late in the game after much hard work has been done. Also, the policy does not develop intelligently with the machine-learning model.

The benefit of reinforcement-learning algorithms is that the final objective function can be the realized/unrealized profit and loss but also values like the Sharpe ratio, maximum drawdown, and value-at-risk measures. Reinforcement learning only has four or so steps as opposed to the seven or eight of supervised learning. RL allows for end-to-end optimization on what maximizes rewards. The RL algorithm directly learns a policy. RL has to take an action in an interactive environment. Compared with supervised learning, which answers the question *Will the asset increase in price tomorrow?* reinforcement learning answers the question *Should I buy the asset today?* The reinforcement-learning algorithm is therefore already packaged as a trading strategy. This does not mean that it is necessarily hard to create a trading strategy out of a supervised-learning task; for example, one can simply buy all assets that are predicted to increase in price tomorrow.

[15] A learning framework that constantly updates the model as new data is made available.

Therefore, the reinforcement learning process draws on a larger process of automation. Similar to supervised strategy development, you still have to ensure that the model works, but here instead of backtesting you use a simulated environment or paper trading. Remember that the focus should remain on out-of-sample performance at the end of the day, so be sure to deflate your performance metrics appropriately to control for multiple testing. In a nutshell, RL comprises data analysis, agents training in a simulated environment, paper trading, and then finally live trading. In each of the last three steps the agent gets exposed to an environment. The simplest RL approach is a discrete action space with three actions: buy, hold, and sell. Unlike supervised models, reinforcement models specify an action as opposed to a prediction; however, the decision masks an underlying prediction.

So, if RL provides all these miraculous benefits, why is it barely used in industry? Well, even though RL can lead to a great strategy in fewer steps with less human involvement, it takes longer to train and is very computationally intensive. RL needs a lot of data, even more so than supervised machine learning. It can also be expensive to test if you cannot reconstruct a good simulated environment. In finance this is generally not a big issue, but it does become an issue when accurate environment feedback is necessary, in which case you might have to revert to the real environment when the simulated environment will not cut it; this can become very expensive. Lastly, the bigger the action space, the harder it is to optimize an RL agent.[16]

It is likely that supervised learning will still rule the pack in the foreseeable future. Supervised learning is already quite flexible, and we should expect to see a lot of innovations to bring the experience of developing strategies closer to that of reinforcement learning without forsaking the benefits of supervised learning. For example, researchers in SL for a long time have looked at embedding policy decisions into SL algorithms. Researchers in finance have also written about creating models that predict the best position sizes and entry and exit points (de Prado 2018), bringing the trading policy and rules closer to the ML model and closer to a form of automated intelligence.

Let us consider a few more disadvantages of reinforcement learning. First, RL's convergence to an

[16] Not to mention the ridiculously slow optimization for continuous as opposed to discrete action space.

optimal value is not guaranteed; the famous Bellman update can only guarantee the optimal value if every state is visited an infinite number of times and every action is tried an infinite amount of times within each state, so essentially never. You of course do not need a truly optimal value; approximate optimality is fine. The big issue is that the sample size, needed to obtain a good level of approximate optimality, increases with the size of the state and action space. Further, without any assumptions there is no better way than to explore the space randomly, so progress at first is small and slow. Continuous states and actions are a serious problem; how are we supposed to visit an infinite number of states an infinite number of times for an infinite number of continuous values with small and slow time steps?

Some of the best approximations can only be done through the generalized nature of supervised learning. Generalization can also be adopted in RL using function approximation as opposed to storing infinite values in an infinitely large table. It is worth nothing that this function approximation is still orders of magnitude harder than normal supervised learning problems, the reason being that you start off the model with no data, and as you collect data the action value changes and the ground truth labels also remain unfixed; a point previously labeled as good, might look bad in the longer run. To get closer to the true function, the agent has to keep exploring. This exploration in uncertain dynamics means that RL is way more sensitive to hyper-parameters and random seeds than SL because it does not train on a fixed data set and is dependent on network output, exploration mechanism, and environment randomness. Thus, the same run can produce different results. But do notice how great it is that you are never given any samples from the true target function, yet you are able to learn by optimizing on a goal. That is why RL is so popular as a concept.

We simultaneously expect to see a lot of improvement on the RL trading front, so that RL adopts the advantages of SL trading methods while not forgoing its own strengths. Conceptually RL offers a kind of paradigm shift where we are not overtly focused on predictive power, which is an auxiliary task, but rather the optimization of actions, which is and has always been the primary goal. SL and RL algorithms indirectly pick up on well-known trading strategies without having to predefine and identify them. For example, the gradient step that leads the machine agent to buy more of what did the best yesterday is indirectly creating a momentum investing strategy. We can expect machine learning to become part of the toolkit of all asset managers in the future.

CONCLUSIONS

Around 40 years ago Richard Dennis and William Eckhardt put systematic trend-following systems on a roll; 15 years later, statistical arbitrage made its way onto the scene; 10 years later, high-frequency trading started to stick its head out. In the meantime, machine-learning tools were introduced to make statistical arbitrage much easier and more accurate. Machine learning today, among other things, assists investment managers to refine the accuracy of their predictions by using supervised learning, improve the quality of their decisions by using reinforcement learning, and enhance their problem discovery skills by using unsupervised learning.

Technological adoption within portfolio management moves fast, and over the decades we have seen technologies come and go. It is likely that this cycle in quantitative finance will persist and that it also applies to machine learning in asset management, with one caveat: Machine learning is also practically revolutionary; instead of just maximizing alpha, it also minimizes overheard costs. Machine learning is already having large economic effects on many financial domains, and it is poised to grow further. Advanced machine-learning models present myriad advantages in flexibility, efficiency, and enhanced prediction quality.

In this article we have paid special attention to how machine learning can be used to improve various types of trading strategies. We started by identifying important components to asset management in the context of machine learning, one of which is portfolio construction, which itself was divided into trading and weight optimization sections. The trading strategies were classified according their respective machine-learning frameworks (i.e., reinforcement, supervised and unsupervised learning). The article finished with a section explaining the difference between reinforcement learning and supervised learning, both conceptually and in relation to their respective advantages and disadvantages. The next article in this series discusses weight optimization strategies.

REFERENCES

de Prado, M. L. 2018. "The 10 Reasons Most Machine Learning Funds Fail." *The Journal of Portfolio Management* 44 (6): 120–133.

Rapach, D. E., J. K. Strauss, J. Tu, and G. Zhou. 2019. "Industry Return Predictability: A Machine Learning Approach." *The Journal of Financial Data Science* 1 (3): 9–28.

ADDITIONAL READING