Machine Learning in Asset Management

Data + Code

July 2019

Derek Snow

Introductory Thoughts

Machine learning (ML) in finance can take on an (1) academic, (2) front office, (3) and a back office form. The techniques used in the front and back office falls within financial industry machine learning[1]. This paper will look at techniques from both an academic[2] and industry machine learning perspective. Quantitative finance offers a wealth of opportunities to learn more about applied and research-driven data science (DS). For finance problems, the goal is easy but the solution is hard. You have vast quarries of data available that can readily be applied using the most recent ML development. Pick your favourite ML/DS acronym; I can assure you there is a place for it in finance. This paper is not meant to be read line by line, it's a *ctr+f* type of affair, let your eyes guide you and follow the blue links for gold.

In most fields, we see a mismatch between industry and academia. That is not the case for machine learning. By nature of its short development cycle, academia and industry closely cooperate frequently and hire out of the same talent pool; this is not just the case in finance but in all applied machine learning fields. Academic finance seems to be applying machine learning differently depending on whether they are in the mathematical finance and traditional finance camp. Within these groups, there is a second division along the lines of predictability and interpretability; the result of regulatory requirements. Financial machine learning is divided into groups that work on the interpretability (b-hatters) and explainability of their models (y-hatters). The b-hatters focus on developing models that would please causal theorists and regulators and the y-hatters side with prediction science and hedge-funds.

B-hatters generally sides with traditional finance and y-hatting with financial mathematics and engineering. This might turn out to be a false dichotomy; in my own research, I have found both to be extremely important for hypothesis and prediction tasks. In this paper, I am looking at the applications of machine learning in asset management, being one of many financial services. Financial services can broadly be divided into, corporate and retail banking, investment banking, insurance services, payment services, investment services, financial management, and other advisory services. In this, paper the focus in on investment services. This paper will pay specific attention to asset management and brokerage services as opposed to wealth management, private equity and venture capital management, not to say that they will not benefit from the same machine learning developments. This paper is conducted in Python; if you do not know Python or machine learning have a quick run through this introduction.

Historically algorithmic trading used to be more narrowly defined as the automation of sell-side trade execution, but since the introduction of more capable algorithms the definition have grown to include idea generation, alpha factor design, asset allocation, position sizing and the testing of strategies. Firms hiring quants as faced with an explore-exploit dilemma when weighing up candidates. As it stands the hiring criteria at most firms are highly in favour of exploit driven abilities, which is unfortunate, because the future of great strategies lies in financial 'hackers'. In industry we see time and time again how applied machine learning enthusiast outperform and outthink machine learning PhDs. We have entered a period of experimentation due to fast computation feedback. We need thinkers in the true original sense of the word.

[1] This paper applies machine learning models from an applied industry finance side as opposed to a business automation side i.e. financial machine learning (FinML) as opposed to business machine learning (BML).

[2] The front office does and will keep on benefiting from both traditional and mathematical academic finance research. Academic research generally require some additional steps to make the research it 'practical'. Both the front and back office also benefits from advances in business machine learning, which is a subsection of AI unrelated to the specific industry but essential for the general automation of work and administrative processes and decision-making.

Asset Management

Each one of finance's subsectors and services will benefit from AI, be it industry or administrative AI. To prove it I will at some point in the future include a few working examples for each of the seven sectors of finance. Asset management forms part of investment services. Asset management can be further broken into the following tasks: portfolio optimisation, risk measures, capital management, infrastructure and deployment, and sales and marketing. The focus of this paper is on portfolio optimisation. Here the term is broadly interpreted to mean trading strategies and weight optimisation methods to construct the optimal portfolio.

1. **Portfolio Optimisation:**
    - Trading Strategies
    - Weight Optimisation
    - Optimal Execution
2. **Risk Measurement:**
    - Extreme Risk
    - Simulation
3. **Capital Management**
    - e.g. Kelly Criterion
4. **Infrastructure and Deployment**
    - Cloud
    - Diagnostics
    - Monitoring and Testing
5. **Sales and Marketing**
    - Product Recommendation
    - Customer Service

Portfolio Optimisation

Trading Strategies

1. Tiny CTA
Resources:
See this paper and blog for further explanation.
Data, Code

2. Tiny RL
Resources:
See this paper and/or blog for further explanation.
Data, Code

3. Tiny VIX CMF
Resources:
Data, Code

4. Quantamental
Resources:
Web-scrapers, Data, Code, Interactive Report.

5. Earnings Surprise
Resources:
Code

6. Bankruptcy Prediction
Resources:
Data, Code, SSRN

7. Filing Outcomes
Resources:
Data

8. Credit Rating Arbitrage
Resources:
Code

9. Factor Investing:
Resources:
Paper, Code, Data

10. Systematic Global Macro
Resources:
Data, Code

11. Mixture Models
Resources:
Data, Code

12. Evolutionary
Resources:
Code

13. Agent Strategy
Resources:
Code

14. Stacked Trading
Resources:
Code, Blog

15. Deep Trading
Resources:Code

Weight Optimisation

1. Online Portfolio Selection (OLPS)
Resources:
Code

2. HRP
Resources:
Data, Code

3. Deep
Resources:
Data, Code, Paper

4. Linear Regression
Resources:
Code, Paper

5. PCA and Hierarchical
Resource:
Code

Other

1. GANVaR
Resources:
Code

All Data and Code

GitHub Project Repository

LinkedIn FirmAI Profile

Trading Strategies

An important first step in constructing an actively managed portfolio is to identify assets and trading strategies that can benefit the investor. In this section trading, all strategies are given a machine learning interpretation and the terminology is used loosely. CTA is a strategy where one takes a position in an asset only after a trend appears in the pricing data. Reinforcement learning strategies uses statistical and machine learning tools to maximise a reward function. Reconstructed asset strategies seeks to uncover arbitrage by reconstructing tradeable assets (derivate) using individual components. Quantamental strategies offers additional information by deriving fair valuations using machine-learning models. Event-driven strategies puts a probability on an outcome occurring and use this probability to consider whether they can identify a viable trading opportunity. Statistical arbitrage seeks mispricing by detecting security relationships and potential anomalies believing the anomaly will return to normal. Factor investing involves the acquisition of assets that exhibit a trait associated with promising investment returns. Systematic global macro, which relies on macroeconomic principles to trade across asset classes and countries. Unsupervised strategies are any strategies that make use of an unsupervised learning method to improve the strategy's prediction accuracy and profitability. Free agents strategies are gradient free reinforcement learning strategies that rely on random movements and introduction of Gaussian movements to select the best performing agents. Gradient agent are reinforcement learning strategies that seek to maximise a reward function though a method known as gradient descent. Supervised learning strategies make price or classification prediction for a point of time in the future that can be used to design trading strategies.

In the examples that follow, I sometimes neglect to include the code, the data, and sometimes both. This is mostly due to copyright and data policy concerns. The strategies in this paper comes from my own work, collaborations and the work of others. In total, there are 15 distinct trading varieties and around 100 trading strategies. Where the work is my own, I try to, as best I can provide for the full pipeline. Note: this is not investing advice - but probably should be.

Tiny CTA

Credit *Man Group*

Momentum refers to the persistence of returns. Winners tend to keep on winning and losers keep on losing. In this momentum strategy, a CTA-momentum signal is built on the crossover signal of exponentially weighted moving averages. One selects three sets of time-scales with each set consisting of short and long exponentially weighted moving averages (EWMA), 2. $S_k = (8,16,32), L_k = (24,48,96)$. Each of these numbers translates in a decay factor that is plugged into the standard definition of EWMA. The half-life is given by:

$$HL = {\log(0.5)}\Big/{\log\left(\frac{n-1}{n}\right)}$$

For each $k = 1,2,3$ one calculate

$$x_k = EWMA[P|S_k] - EWMA[P|L_k]$$

The next step is to normalise with a moving standard deviation as a measure of the realised 3-months normal volatility (PW=63)

$$y_k = \frac{x_k}{Run.StDev[P|PW]}$$

The series is normalised with the realised standard deviation over the short window (SW=252)

$$z_k = \frac{y_k}{Run.StDev[y_k|PW]}$$

Next one calculate an intermediate signal for each $k = 1,2,3$ via a response function R

$$\begin{cases} u_k = R(z_k) \\ R(x) = \dfrac{xexp(^{-x^2}/_4)}{0.89} \end{cases}$$

Then the final CTA momentum signal is the weighted sum of the intermediate signal where we have chosen equal weights, $w_k = \frac{1}{3}$

$$S_{CTA} = \sum_{K=1}^{3} w_k u_k$$

For the first strategy, I did not include any actual machine learning adjustments, but I chose an example where it is extremely easy to add. By identifying all the arbitrarily chosen parameters like $S_k, L_{k,} and\ w_k$ and optimising them using a reinforcement-learning algorithm one can venture to improve the Sharpe ratio or simply the returns. See the next strategy for an implementation of a tiny reinforcement-learning algorithm and see if you can apply the framework to this example.

Resources:

See this paper and blog for further explanation.

Data, Code

Tiny RL

Credit Teddy Koker

In this example, we will make use of gradient descent to maximise a reward function. The Sharpe ratio will be the chosen metric for the reward function and can be defined 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:

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

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.

$$F_t = tanh(\theta^T x_t)$$

The input vector is $x_t = [1, r_{t-M}, \dots, F_{t-1}]$ 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 is can use to calculate the next position. Once we have a position at each time step, we can calculate our returns R at every step using the following formula. In this example, δ is the symbol for transaction cost.

$$R_t = F_{t-1} r_t - \delta |F_t - F_{t-1}|$$

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 above formula. It can be written follows:

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

Resources:

See this paper and/or blog for further explanation.

Data, Code

Tiny VIX CMF

With Andrew Papanicolaou

VIX and futures on VSTOXX are liquid and so are ETNs/ETFs on VIX and VSTOXX. Prior research shows that the curves exhibit stationary behaviour with mean reversion toward a contango. It is known that one can imitate the futures curves and ETN price histories by building a model and then use that 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, $V_t^\theta = F_{t,t+0}$, for $t \le T_1 \le t + \theta \le T_2$ define

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

And note that:

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

The CMF is the interpolation,

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

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 simply want to focus on maturity and instrument selection, and therefore ignored the roll yield and simply focused on the CMFs value. However, if you are interested, the value of the ETN can be obtained as follows.

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

Where *r* is the interest rate.

Unlike the previous 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 that results in a matrix of $2^7 - 2 = 126$ combinations. Then use a standard normal distribution to randomly assign weights to each item in the matrix. Create an inverse matrix and do the same. Now normalise the matrix so that each row equals one in order 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 as this whole trading strategy is serialised. 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* percent of strategies for their highest median cumulative sum over the period. Of that selection, select the top *Y* percent for the lowest standard deviation. Of that group, select *Z* percent again from 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. Of the remaining strategies, iteratively remove high correlated strategies until only 10 (*S*) strategies remain. With that remaining 10 strategies, which have all been selected using only training data, use training data again to formulise a reinforcement learning strategy using a simple MLP neural network with two hidden layers. 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.

Resources:

Data, Code

Quantamental

In my experience, Quantamental strategies are intelligent data driven valuations. A good example of a machine learning Quantamental strategy is a project I worked on in late 2017. The core goal of the project was to estimate a fair valuation of privately owned restaurant chains. A secondary consequence of the project was the ability to predict whether a publicly traded restaurant was under or overvalued. To do this alternative data was gathered at an employee, customer, shareholder and management level for the parent companies and where possible the individual locations. In the process, data was gathered from more than ten alternative sources using web-scrapers. This includes LinkedIn, Yelp, Facebook, Instagram and Glassdoor data. What followed was the use of an open sourced gradient boosting model from Yandex known as Catboost. In my final implementation I preferred a model called XGBoost, this is not shown in the code. I give the mathematical definition of the XGBoost model in the appendix. Both of these models are gradient boosting models (GBMs).

Here is how one might come to understand a GBM model. The algorithms for regression and classification only differ in the loss function used and is otherwise the same. To create a GBM model we have to establish a loss function, *L* to minimise, to optimise the structure and performance of the model. This function has to be differentiable, as we want to perform a process of steepest descent, which is an iterative process of attempting to reach the global minimum of a loss function by going down the slope until there is no more room to move closer to the minimum. For the regression task, we will minimise the mean squared error (MSE) loss function. The focus here is on $f(\boldsymbol{x}_i)$ as this is the compressed form of the predictor of each tree *i*.

$$L(\theta) = \sum_i (\boldsymbol{y_i} - f(x_i))^2$$

$$L(\theta) = \sum_i (\boldsymbol{y}_i - \hat{y}_i)^2$$

Further, it is necessary to minimise the loss over all the points in the sample, $(\boldsymbol{x}_i, y_i)$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} L(\theta)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$$

At this point we are in the position to minimise the predictor function, $f(\boldsymbol{x}_i)$, w.r.t. x since we want a predictor that minimises the total loss of $f(\boldsymbol{x})$. Here, we simply apply the iterative process of steepest descent. The minimisation is done in a few phases. The first process starts with adding the first and then successive trees. Adding a tree emulates adding a gradient based correction. Making use of trees ensures that the generation of the gradient expression is successful, as we need the gradient for an unseen test point at each iteration, as part of the calculation $f(\boldsymbol{x})$. Finally, this process will return $f(\boldsymbol{x})$ with weighted parameters. The detailed design of the predictor, $f(\boldsymbol{x})$, is outside the purpose of the study, but for more extensive computational workings see the appendix.

Among other things, the algorithm predicted that BJ's restaurants market value was trading about 40% below its competitors with similar characteristics; within the year the algorithm was proven right and the price just about doubled compared to its model-defined competitors. This project was especially interesting because no company specific financial data was used as inputs to the model, and the response variable was simply the market value. In the process, I also developed an interactive report that is now open sourced. If you have a look at the report, the light blue line signifies what the 'AI' recommends the cumulative return should have been after 5 years, whereas the dark blue line is the cumulative return over time for BJ's; being about 40 percentage points lower than what the AI believed to be a 'fair' market value. I am not going to make everything open for this project, but the links below would give you some crumbs[i].

Resources:

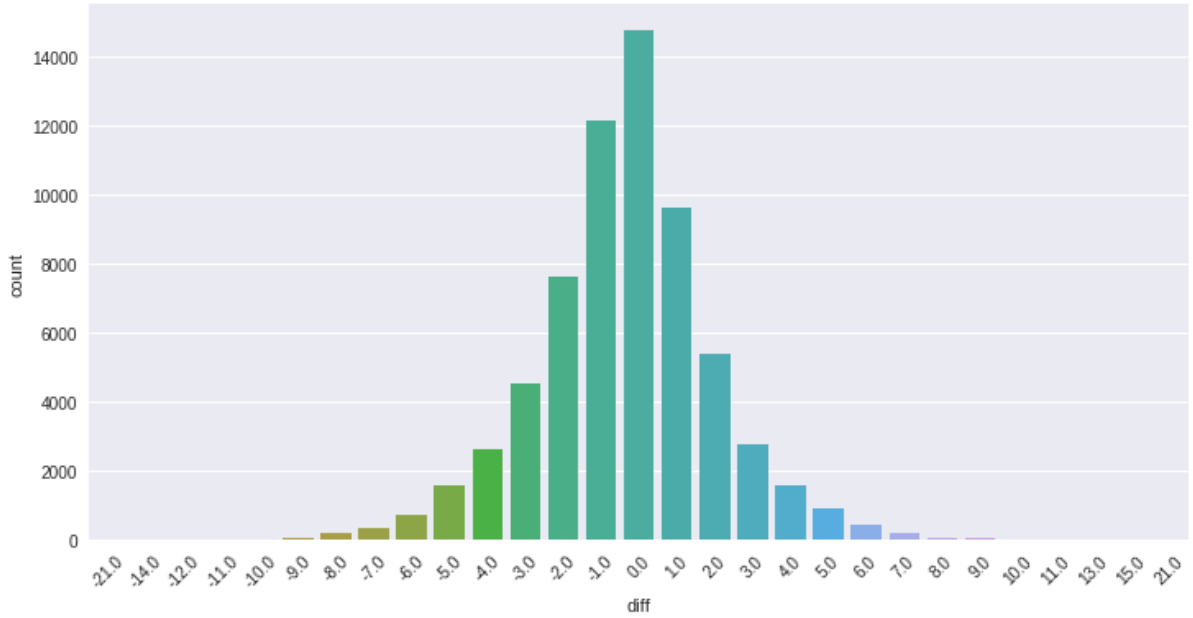
Web-scrapers, Data, Code, Interactive Report, Paper

Credit Rating Arbitrage

Three years ago, I was particularly interest in agency credit ratings. As one does, I developed a prediction model to come to grips with the issues of the credit rating industry. The process involved the training of an XGBoost model to predict companies' credit ratings and then cross-sectionally applying prediction to new companies to remove time bias and investigate rating discrepancies. In other words, the companies' actual rating are compared against their model predicted ratings; which leaves you with firms that have deviated the most from learned patterns.

With this model and prediction in place, you can hypothesise that the divergence is either a true reflection of a rating change to come, in which case you can establish a long-short portfolio to short positive and long negative deviations and the literature clearly shows that a rating change has an impact on the price of the company's stock. This did not produce great results. Lastly, one can simply call the difference soft credit risk – that which cannot be picked up by financials (the only input data used), and then use that as another type of risk indicator for firms. This post-hypothesis did end up working well. I haven't formulised this theory and I am happy for anyone to pick this project up and run with it.

The model has been shown to perform equivalent to that of a rating agency in predicting defaults. This leads one to believe that rating agencies truly suffer from some revenue and other incentive biases given that the model is fed only a small amount of data compared to that available to the agencies. With this model in hand one can easily create a free only rating agency whereby you certify peoples ratings using your models, all you need is 4 years financial data. With this model, unlike the agency ratings, it is also possible to assign rating intervals as opposed to punctual ratings.



Resources:

Code

Event Driven Arbitrage

Earnings Surprise

In this section, I will describe two event driven strategies; the first is an earnings prediction strategy that investigates the potential of predicting an earnings surprise and the second is a bankruptcy prediction strategy. The earnings surprise strategy starts of as a simple classification task where the response variable is the occurrence of an earnings surprise. An earnings surprise is defined as a percentage change from the analyst's EPS expectation as described in the data section and the actual EPS as reported by the firm. In this study, we include percentage thresholds, *s*, as a means of expressing the magnitude of a surprise to construct various tests. This trading strategy works well historically, but the performance is slightly waning over the years.

$SURP_{itsx} = 1 \rightarrow$ Neutral;

$SURP_{itsx} = 2,\ where\ \ \frac{EPSAC_{it} - EPSAN_{it}}{EPSAN_{it}} - 1 > s \rightarrow$ Positive;

$SURP_{itsx} = 0,\ where\ \ \frac{EPSAC_{it} - EPSAN_{it}}{EPSAN_{it}} - 1 < -s \rightarrow$ Negative.

To provide some clarity, *i* is the firms in the sample; *t* is the time of the quarterly earnings announcement; *s* is the respective constant surprise threshold, 5%, 10% or 15%; *x* is a constant percentage of the sample selected sorted by date of earnings announcement; *EPSAN* is the earnings per share forecast of analysts and *EPSAC* is the actual earning per share measure as reported by the firm. The surprise measure is simply the difference between the actual and expected EPS scaled by the expected EPS.

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

$(\mathbf{1})\ TrainedModel = ModelChoice;\ (\mathbf{2})\ Predictions = TrainedModel(TestInputs);\quad (\mathbf{3})\ Metrics = Functions(TestTarget, Predictions).$

And more specifically a classification task using and XGBoost GBM can be presented as follows,

$(\mathbf{1})\ Classifier = XGBoostTreeClassifier\big(Train_X, SURP_{its(x)}, Paramaters\big);\ (\mathbf{2})\ PredSURP_{ith} = Model(Test_X);\ (\mathbf{3})\ Metrics = Functions\big(SURP_{its(1-x)}, PredSURP_{ith}\big).$

The prediction values, $PredSURP_{ith}$, of the classifier is a categorical variable that falls within the values $\{0,1,2\} \rightarrow \{\text{Negative Surprise, No Surprise, Positive Surprise}\}$, for surprises of different thresholds, $\text{h}\ \{5\%, 10\%, 15\%\}$. If we assume that the training set is 60% of the original data set, then the training set's target value is $SURP_{its\ (60\%)}$, being the first 60% of the dataset ordered by date. As a consequence, the test set's target values are $SURP_{its\ (40\%)}$, the last 40% of the dataset. The metrics for a classification task comprise of accuracy (proportion of correctly classified observations), precision (positive predictive value), recall (true positive rate), and confusion matrices/contingency tables

The following expresses a simple strategy by going long on stocks that are expected to experience a positive 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, as there is on average only four firms in a portfolio of expected surprises but there can be as few as one firm in a portfolio[3]. Earnings surprises are often termed as soft events in books on event driven investment strategies. Event driven investment forms a large part of the hedge fund industry, accounting for about 30% of all $2.9 trillion assets under management according to Hedge Fund Research 2015. Event driven strategies are a great way to benefit from increased volatility. The strategies recommended in this section fully invests capital in each event, it is therefore smart to include some sort of loss minimisation strategy.

As a result, the first strategy incorporates a stop loss for stocks that fell more than 10%; 10% is only the trigger, and a conservative loss of 20% is used to simulate slippage. This is done by comparing the opening with the low price of the day. There are an endless number of opportunities and strategies that can be created; it is, therefore, important to select simple strategies not to undermine the robustness of this study. In saying that the choice of slippage is not backed by any literature and is an arbitrary, albeit conservative choice to the strategy. I have created three different portfolios consisting of firms that are expected to experience a positive surprise above a 5%, 10%., and 15% threshold. P = $\{P_5, P_{15}, P_{30}\}$. What follows is a simple return calculation for an earnings announcement

[3] For robustness, I have also tested for value-weighted returns which showed a slight increase in improvement.

of firm *i* at time *t*, where the daily low price, Pl_{ti}, is not more than -10% lower than the closing price $P_{(t-1)i}$. If it is, then a slippage loss of -20% is allocated to the return quarter of that firm.

$$R_{jt} = \frac{(P_{ti} - P_{(t-1)i})}{P_{(t-1)i}}, \qquad if \ \frac{Pl_{ti} - P_{(t-1)i}}{P_{(t-1)i}} > -10\%, \ R_{jt} = -20\%$$

In this equation, i is the firms that have been predicted to experience an earnings surprise at *t*. $P_{(t-1)i}$ is the closing price of the common stock of firm *i* on date *t-1*. Pl_{ti} is the daily low of the common stock price of firm *i* on date *t-1*. The equal weighted return of a portfolio of surprise firms is then calculated as so,

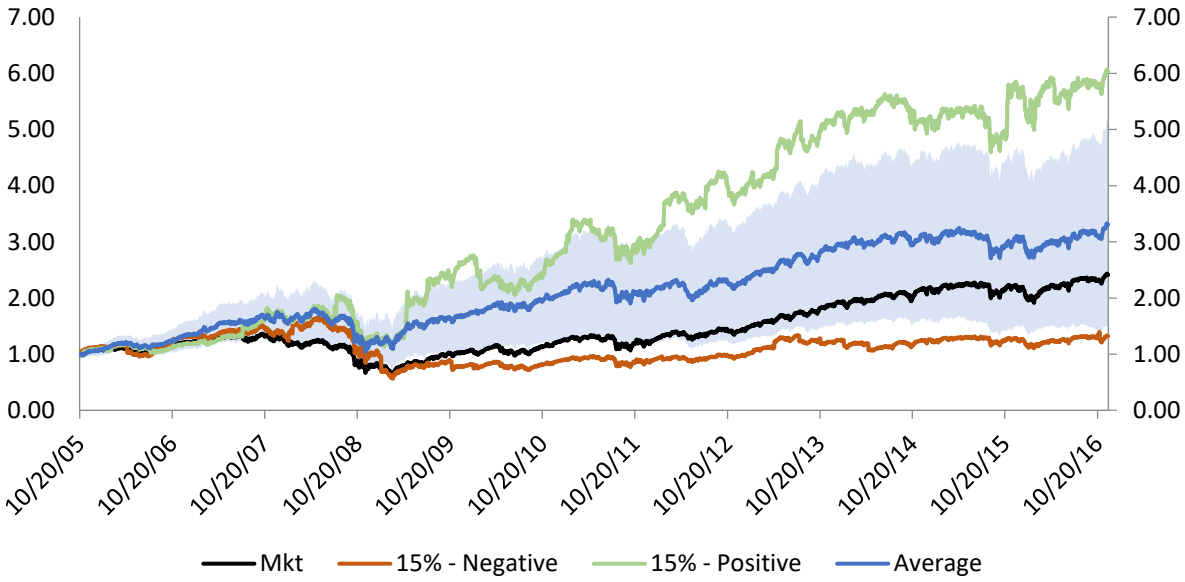
$$R_{pt} = \frac{1}{n}\sum_{j=1}^{n_{pt}} R_{jt} \qquad (1)$$

In this equation, j is all the firms that experience surprises on date t. R_{it} is the return on the common stock of firm *j* on date *t*. n_{pt} is the number of firms in portfolio *p* at the close of the trading on date *t-1*. The equation below is the five-factor asset-pricing model. In this equation, R_{pt} is the return on portfolio p for period t, that has been predicted to experience an earnings surprise event at *t*. R_{Ft} is the risk-free rate. R_{Mt} is the value-weighted return of the market portfolio. SMB_t, HML_t, RMW_t and CMA_t are the respective differences between diversified portfolios of small stocks and big stocks, high and low B/M stocks, robust and weak profitability stocks, and low and high investment stocks. To perform the regressions, the respective daily values were obtained from Kenneth French's website[4].

$$R_{pt} - RF_t = a_i + bi\,(R_{Mt} - R_{Ft}) + s_i SMB_t + h_i HML_t + r_i RMW_t + c_i CMA_t + e$$

$$R_{cum} = \prod_{t=1}^{m}(1 + R_{pt}) - 1$$

Figure 1: Portfolio Value - Large Firms 15% Surprise Prediction Strategies



[4] http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

This portfolio reports the cumulative returns of a buying and holding positive and negative surprise portfolios for all firms in the sample with a market value of $10 Billion. On average, there are about four firms for each portfolio day. On days where no trading surprises occur position in the market is taken. The band in the middle is the significance band obtained by a Monte Carlo simulation from randomly predicting and taking a position in 774 firms before an earnings announcement. The chart also reports the cumulative portfolio return of the market as calculated from the market returns obtained from French's website. The chart shows that negative surprises, pretty much tracks the market portfolio. It is possible that some return can be earned by shorting these surprises over certain periods, but on average it is not a very profitable strategy due to the small amount of shorting opportunities. In total, there is 2944 trading days, for the long strategy, 215 of these days are returns from earnings surprises comprising 774 firms and for the short strategy 62 days comprising 234 firms.

Resources:

Code, Paper

Bankruptcy Prediction

The bankruptcy task involved the acquisition of data that includes all corporate bankruptcies in the US. A bankruptcy prediction model was created to predict these bankruptcies using only standardised accounting information, one and two years before the bankruptcy occurs. The table below reports on the outcome of the model by means of a confusion matrix.

Table 1: Healthy and Bankrupt Confusion Matrix.

Aggregated Health and Bankrupt Firms Matrix		**Predicted**		Sample Proportion
		Healthy	Bankrupt	
Actual	Healthy	**29041 - TN**	116 - FP	0.96
	Bankrupt	805 - FN	**258 - TP**	0.03
Precision		0.97	0.69	30220
Improvement		0.01	0.66	-

This bankruptcy prediction task solves for a binary classification problem that produces a 2×2 matrix. The columns of the matrix represent the predicted values, and the rows represent the actual values for bankrupt and healthy firm predictions. In the cross-section of the rows and columns, we have the True Positive (TP), False Negative (FN - type II error), False Positive (FP - type I error)), and True Negative (TN) values. The sample proportion on the far right is equal to all the actual observations of a certain classification divided by all the observations. The *precision* is calculated by dividing the true positives (Bankruptcies) with the sum of itself and the false negatives (Healthy). An example along the second column: $258/(116 + 258) = 69\%$. The improvement is the percentage point improvement the prediction model has over a random choice benchmark.

The average ROC (AUC) of more than ten *past* decision tree ensemble studies (in literature: 2018) is around 0.927. The best performing is 0.997 and the worst performing is 0.859. In spite of the conservative sample selection in this chapter, the decision tree ensemble (XGBoost) model used in this study performed better than the average of past reported studies. It is also the best model when compared to other studies that only used accounting values as inputs. The average AUC of eight

different neural network studies is 0.850, the best and worst performing past study has an AUC of 0.901 and 0.750 respectively. The DCNN of this model achieved an AUC of 0.9142, making it the best performing neural network of all past research.

Table 2: Model Comparison Using Different Inputs

Metrics	All Data	50 Variables Model	One Year Before Bankruptcy	Two Years Before Bankruptcy
ROC AUC Sore	0.9587	0.9408***	0.9666**	0.9434***
Accuracy Score	0.9755	0.9700	0.9860	0.9837
False Positive Rate	0.0037	0.0056	0.0010	0.0002
Average Log Likelihood	0.1414	0.1795	0.1682	0.2206

This table compares the performance of model that includes only 50 of the most predictive variables as inputs, a model that only includes bankruptcy observations one or two years before the filing. All statistical comparisons are made against the model called "All Data." *p<.1 ** p<.05 *** p<.01. Significance levels are based on a two-tailed Z test.

Resources:

Data, Code, Paper

Filing Outcomes

Following on from the bankruptcy prediction strategy, this study further contends that past research's black-and-white view of solely predicting the occurrence of a legal bankruptcy is not sufficient; the reason being that the economic effect of the outcome is largely determined by the characteristics associated with the bankruptcy. Filing outcomes have great economic consequence for creditors and shareholders. Stakeholders would want to know the likelihood of a litigated bankruptcy occurring as well as the likely filing outcomes associated with the bankruptcy.

Binary Classification Performance for Predicting Bankruptcy Outcomes.

Binary Classification Model	ROC AUC Sore	Accuracy Score	Average Log Likelihood	Average Precision Score	False Positive Rate	False Negative Rate
Duration	0.62	0.56	0.67	0.69	0.66	0.26
Survival	0.73	0.69	0.59	0.80	0.61	0.12
Chapter	0.88	0.95	0.50	0.70	0.05	0.20
Asset Sale	0.64	0.66	0.61	0.39	0.27	0.55
Tort	0.54	0.90	0.40	0.17	0.05	0.83

This table reports six important metrics for five alternative classification tests to predict the outcome of predicted bankruptcies. *Duration* classification is the first task to predict the binary outcome. This task involves the prediction of whether or not the disposition will take longer than a year after the initial filing. *Survival* predicts a binary outcome as to whether or a firm will re-emerge out of bankruptcy and remain in business for longer than 5 years. The *Chapter* task predicts whether the bankruptcy filing would be converted to Chapter 7 or whether it will be a Chapter 11 filing. The *Asset Sale* model predicts whether the debtor will sell all or substantially all the assets during the Chapter 11 proceedings. The *Tort* classification task seeks to predict whether the bankruptcy would occur as a result of tortious actions such as product liability, fraud, pension, environmental and patent infringement claims. The above metrics have been fully defined in table X.

The prediction of all filing outcomes is contingent on a correctly predicted bankruptcy outcome. All these models use simple accounting value inputs the year before the filing date. Filing outcomes have great economic consequence for creditors and shareholders. Stakeholders would want to know the likelihood of a litigated bankruptcy occurring as well as the likely filing outcomes associated with the bankruptcy. In the table below, I present the performance of five different filing outcome models.

The first of these five is the *chapter* prediction model. It involves a prediction task of whether the bankruptcy will finally be filed under chapter 7 or chapter 11. The chapter prediction model performed the best of all other filing outcomes models. It achieved an AUC of 0.88. The survival prediction model that identifies whether the firm would emerge from bankruptcy performed second best with an AUC of 0.73. The prediction task that attempts to predict whether assets will be sold in a 363 Asset sale or by other means, came in third with an AUC of 0.64. The duration task, which involves the prediction of whether the bankruptcy proceedings would endure for longer than one year came in second to last with an AUC of 0.62. And lastly, the tort task had an AUC score of 0.54 which is only slightly higher than random. All prediction tasks performed better than random guessing.

Resources:

Data

Statistical Arbitrage:

Pairs trading is a simple statistical arbitrage strategy that involves finding pairs that exhibit similar historic price behaviour and then to, once they diverge, bet on the expectation that they will convergence. In a universe of assets over time, $\boldsymbol{X}_t$, pick individual 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[ii] 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 can't have drift. Z_t is thus the pairs position, 1 share long in X_t^1, 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. We can specify an easy and arbitrary trading rule, which specifies that when $abs(X_t^1 - K_0 X_t^2)/(\frac{1}{n}\sum_{t=1}^{n} abs(X_t^1 - K_0 X_t^2)) > 0.1$, you sell the highly priced asset and buy the lowly priced asset, and do the reverse as soon as the above formula hits 0.5; at which point 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 the above. Instead of using simple trading rules, dynamic model based approaches can be used to define the trading logic. The before mentioned mean reversion process can be discretised into an AR process where the parameters can be estimated iteratively and used in the trading rule.

Researchers can do pair selection using distance, rolling OLS cointegration, or Kalman filter cointegration. Unsupervised learning methods like, hierarchical clustering, k-means, VAE embedding or DBSCAN have also been used with mixed success. Although unsupervised learning methods might not be the best method to directly select pairs from, it is a good intermediate step to create clusters out of which traditional pairs selection strategies can be used. For example, see the following notebook from Yicheng Wang for the available code (data not provided) that uses DBSCAN and PCA. Finally in the last step one can make use of an RL agent to identify the best enter and exit opportunities, i.e., using an RL agent to define the trading rules. I am yet to connect unsupervised learning with reinforcement learning. I would leave it out there for others to do. I have developed a model here, without any strategy attached. If I do come back to this project, I would not use this model. I would start from scratch, building the unsupervised step into the strategy from the start. If you are even more adventurous, have a look at the following code and data for an options arbitrage (code, data) and treasury future trading strategies (code, data; by Raphael Douady), and see if you are able to identify how you would be able to include a reinforcement-learning agent.

Factor Investing

Instead of doing normal factor analysis, we will performing industry factor analysis. In this example, we will use machine learning tools to analyse industry return predictability based on the lagged industry returns across the economy. Annualised return on a strategy that long the highest and short the lowest predicted returns, returns an alpha of 8%. In this approach, one has to be careful for multiple testing and post-selection bias. A LASSO regression is eventually used in a machine-learning format to weight industry importance. The following formulates a standard predictive regression framework:

$$\boldsymbol{y_i} = a_i^* \boldsymbol{\gamma_T} + \boldsymbol{X b_i^*} + \boldsymbol{\varepsilon_i} \quad for\ i = 1, \dots, N,$$

Where

$$y = \left[r_{i,1} \dots r_{i,T}\right] \ ; \ X = \ [x_1 \dots x_N] \ ; x_j = \ \left[r_{i,0} \dots r_{i,T-1}\right] \ for\ j = 1, \dots, N$$

$$b_i^* = \left[b_{i,1}^* \dots b_{i,N}^*\right] \ ; \ \varepsilon_i = \ \left[\varepsilon_{i,1} \dots \varepsilon_{i,T}\right]$$

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

$$\underset{a_1 \in \boldsymbol{R}, b_i \in R^N}{\arg min} \left(\frac{1}{2T} \left|\left| \mathrm{y_i} - a_i \boldsymbol{\gamma_T} - \boldsymbol{X b_i} \right|\right|_{\boldsymbol{2}}^{\boldsymbol{2}} + \vartheta_i \left|\left| \boldsymbol{b_i} \right|\right|_{\boldsymbol{1}}\right)$$

The LASSO generally performs well in selecting the most relevant predictor variables. Some argue that its penalty term over shrinks the coefficient for the selected predictors. In that scenario, one can use the selected predictors and re-estimate the coefficients using OLS. This OLS sub model can be replaced by any other machine learning regressor. In fact the main and sub model can both be machine learning regressors, the first selecting features and second predicting the response variable based on those features.

For additional practice, you can have a look at the following repositories and see if you can identify any machine learning use cases. Factor analysis: 1. Mutual fund factors (data), 2. Equity factors (data), 3. Penalised factors.

Resources:

Paper, Code, Data

Systematic Global Macro

When oil exists a bear market then the currency of oil producing nations would also rebound. In this strategy, we will investigate the effect the price of oil has on the Norwegian Krone and identify whether a profitable trading strategy can be executed. First, one needs a stabiliser currency to regress against. The currency should be unrelated to the currency used in the target. Something like the JPY is a good candidate for the Norwegian Krone. One can then use price of other currencies and Brent as measured against JPY to identify whether the Norwegian currency is under or overvalued. I will use an elastic net as the machine learning technique. It is a good tool when multicollinearity is of issue.

An elastic net is a regularised regression method that combined both L1 and L2 penalties. The estimates from the elastic net method are defined by

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

The loss function becomes strongly convex as a result of the quadratic penalty term therefore providing a unique minimum point. Now that the predictors are in place, one have 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 deviation. At that point, one can expect our interpretation of the underlying model to be wrong.

The elastic net has also usefully been applied in portfolio optimisation. Here, I have applied the two other well-known regularised linear models to portfolio optimisation.

Resources:

Data, Code

Unsupervised Strategy

Mixture Models

Credit: Brian Christopher.

The evidence shows that asset returns violate all the rules of stationarity. For a time series to be stationary it should have a constant mean, variance and covariance with leading or lagged values and the SPY returns exhibit nonstationarity in the mean, variance and demonstrate periods of variable autocorrelation. You can plot the data to see if the mean and variance is constant and do the Dicky Fuller test for a unit root.

As a result we should expect predictions approximated by normal distribution measures like mean and variance to badly predict the future return distribution. Luckily this can also be tested like stationarity that be tested. In fact there is a test called the Komogorov-Smirnov 2 sample test that can be used to compare a test and predicted distribution. It provides a small p-value if there are strong evidence that the two data samples came from different distributions. However, the test is more commonly used to test the test and training samples themselves, which is what is done in the notebook. The results from the test shows that 70% of the train test splits reject the null hypothesis that the training and test data came from the same distribution. As a result, we need a modelling framework that can overcome

these issues. Techniques that cannot accommodate the time varying properties of financial asset returns present a risk to our financial assets.

GMM model can help overcome some of the times series prediction issues as it as an approach to approximate nonstationary distributions. Our biggest issue is that asset returns seems to be comprised from multiple distributions (regimes or states). Each regime has its own composition of characteristics like stable, risky, low and high volatility. If we argue that there is two regimes, stable and risk, we sit with the issue that we don't know the mean and variance parameters for the regimes, because we invariably don't know which datapoint comes from which regime. Fortunately, there is a solution to this problem, the devised solution is an expectation-maximisation algorithm. The expectation-maximisation algorithm underpins many unsupervised learning methods including that of mixture modelling. It has use cases for when data is corrupted, missing or the parameters of the data generating process is unknown and lastly where it is not known which data generating process generated which data point.

You start by guessing the parameters for the two different gaussian distributions you believe the time series exhibits. Assuming you are correct, you assign probability weights to each datapoint from both of the regimes. Then in an iterative process, we normalise the probabilities and estimate the parameter means and volatilities of the regimes. With this approach, we are guaranteed to improve the estimate at each iteration. $E(z_{ij})$ is the probability that x_i was drawn from the jth distribution.

$$E(z_{ij}) = \frac{p(x = x_i|\mu = \mu_j)}{\sum_n^2 p(x = x_i|\mu = \mu_j)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j)^2}}{\sum_{n=1}^2 e^{-\frac{1}{2\sigma^2}(x_i-\mu_n)^2}}$$

The formula simply states that the expected value for z_{ij} is the probability x_i given μ_j divided by the sum of the probabilities that x_i belonged to each μ. After calculating all expected values $E(z_{ij})$ we can update the new μ values.

$$u_j = \frac{\sum_{i=1}^m E(z_{ij})x_i}{\sum_{i=1}^m E(z_{ij})}$$

This formula generates the maximum likelihood estimate. By repeating the expectation and maximisation steps, we are guaranteed to find a local maximum giving us the maximum likelihood estimation of our hypothesis. One question that has to be resolved is the number of components or regimes to select. Here we can make use of AIC or IC criterion to compare the relative suitability of different models. With the GMM model, we can sample directly from the posterior to generate new samples.

Now that we know what the model is, how do we apply it? Well one can use it for a range of tasks like classifying returns as outliers for filtering and data cleaning. One can use it in risk management to filter position sizes, trades, veto trades. For example if we predict a volatile regime going forward, we might want to reduce potion sizing.

In this example, I specifically use GMMs to predict return distributions. We can use it to predict iteratively the distribution of our lookahead period. What the results in this study show is that the accuracy of the return distribution is better with less components. This leads one to question the use of a mixture model. In a further step, we can use a mixture model to implement an event study on

post outlier returns. The analysis appears to show that there is a tradeable pattern based on our filtering conditions.

Resources:

Data, Code

Free Agent

Evolutionary:

A large portion of reinforcement learning algorithms is optimised using Q-learning or its deep learning equivalent, Deep-Q learning. There are 'better' algorithms out there such as policy gradient and its variants like the Actor-Critic method. Although the Actor-Critic model performs well, it can take a very long time to train. The reason reinforcement learning is so slow because the gradients are hard to find. Unlike supervised learning, there is no clear gradient direction for each of the parameters in the network. The gradient information appears occasionally when the environment gives a reward (penalty). Most of the time the agent takes an action without knowing whether it is useful or not. The reinforcement learning algorithm will be on a low error high reward surface and thus are more likely to be initialised on a flat surface where it is hard to move towards better performance. It is not presented with informative reward and penalty signals. In effect, the agent more or less performs a random-walk for a long unproductive amount of tie. For some problems, it might feel like the policy-gradient approach is no better than random search.

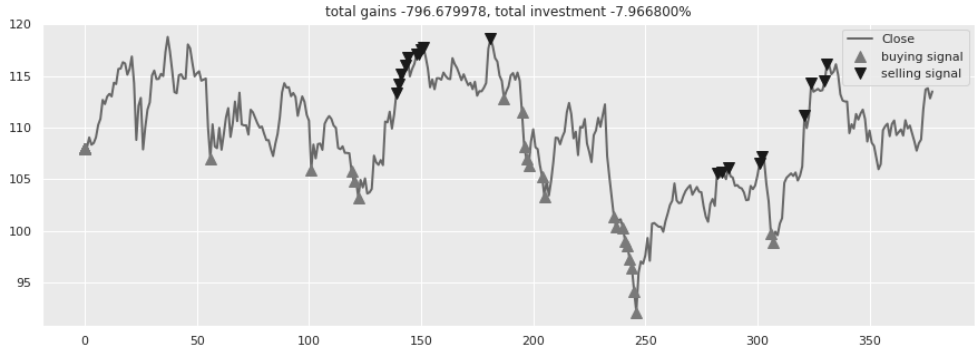
We can look to gradient-free approaches to reinforcement learning for help. Using genetic algorithms one can find the parameters that best defines a good performing agent. You start with a certain number of predefined agents (i.e. potential trading strategies) with randomly initialised parameters. Some of them by chance will outperform others. Here is where the evolution comes to play and resurrects the concept of the 'survival of the fittest'. The algorithm selects the top performing decile of agents and add a bit of Gaussian noise to the parameters so that in the next iteration the agent gets to explore the neighbouring space to identify even better performing strategies. It is down to the core a very simple concept. One can further improve these results by changing some of the input parameters to the genetic algorithm for example the hyperparameters like the window size, population size, variance, and learning rate. To do this one can make use of Bayesian optimisation. In the notebook there are two models, the first is a simple evolutionary strategy agent and the second makes use of Bayesian optimisation.

Resources:

Code and Data

Gradient Agent

In this notebook, 24 agents are developed using different frameworks, the first three do not make use of reinforcement learning; their rules are determined by arbitrary inputs. This includes a turtle-trading agent, and moving-average agent and a signal-rolling agent. The rest of the notebook contains progressively more 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 initialised, trained and measured for performance. Each agent individually generates a chart that contains some of their performance metrics.



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

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

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

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

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

Q-learning: is an online action-value function learning with and exploration policy, e.g., epsilon-greedy. You take an action, observe, maximise, adjust policy and do it all again.

Take some action a_i and observe $(s_i, a_{,i}, s_i', r_i)$

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

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

Then explore with the epsilon-greedy policy:

$$\pi(a_t|a_t) = \begin{cases} 1 - \epsilon \; if \; a_t = argmax_{a_t} Q_\theta(s_t, a_t) \\ \epsilon/(|A| - 1) \; otherwise \end{cases}$$

Policy Gradients: here you maximise the rewards by taking actions with where higher rewards are more likely.

Sample $\{\tau^I\}$ from $\pi_\theta(a_t|s_t)$

$$\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta log \pi_\theta(a_t^i|s_t^i))(\sum_t r(s_t^i, a_t^i))$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

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

$$\text{Take action } a \sim \pi_\theta(a|s) \text{ , get } (s, a, s', r)$$

$$\text{Update } \widehat{V_\emptyset^\pi} \text{ using target } r + \gamma \widehat{V_\emptyset^\pi}(s')$$

$$\text{Evaluate } \widehat{A^\pi}(s,a) = r(s,a) + \gamma \widehat{V_\emptyset^\pi}(s') - \widehat{V_\emptyset^\pi(s)}$$

$$\nabla_\theta J(\theta) \approx \nabla_\theta log \pi_\theta(a|s) \widehat{A^\pi}(s,a)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Resources:

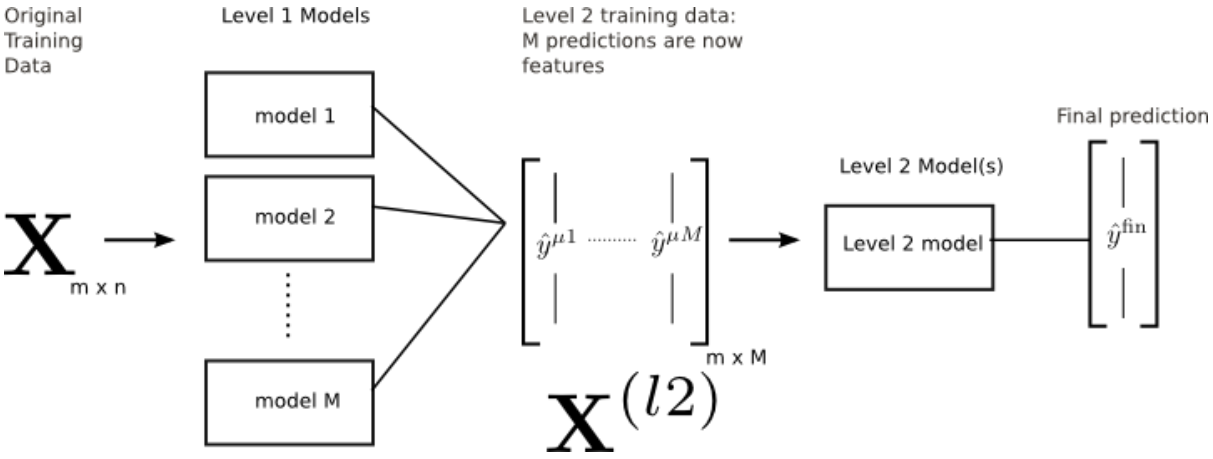
Code and Data

Supervised Learning:

I have not turned any of the supervised learning methods into trading strategies yet. Here I am 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. You can construct the trading policies by hand or rely on reinforcement learning strategies to 'develop' the best trading policies.

Stacked Trading

This 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 pricing stock variables. The following diagram explains the concept of stacking.



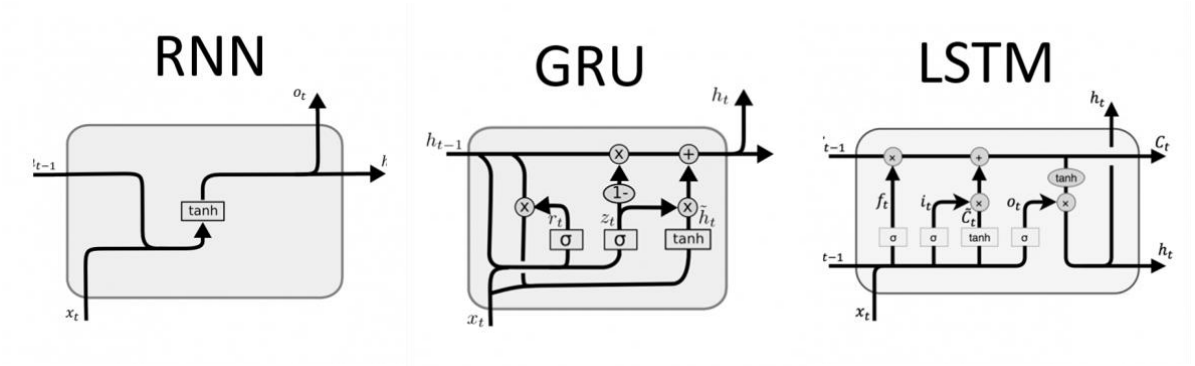
Resources:

Code and Data, Blog

Deep Trading

In this notebook, there are 30 different neural network frameworks. This includes Vanilla RNN, GRU, LSTM, Attention, DNC, Byte-net, Fairseq, and CNN neural networks. The mathematics of the different frameworks are vast and would take too much space to include here, I will therefore refer you to a book and a course.

To understand the major differences consider the following. 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 Gated Recurrent Unit (GRU) further introduces the concept of a gate that decides whether to pass a previous output, (h_{t-1}) to a next cell or not. It is simply an additional mathematical operation performed on the same inputs. With the Long Short Term Memory Unit (LSTM) two additional gates are introduced. In addition to the GRU's update-gate a *forget* and *output* gate are included. Again, these are additional mathematical operations on the same inputs.



Resources:

Code and Data

Weight Optimisation

Online Portfolio Selection (OLPS)

Online Portfolio Selection (OLPS) sequentially selects a portfolio over a set of assets in order to achieve a certain target such as maximising cumulative wealth. In this analysis, we can use ETFs to avoid survivorship bias. State of the art online portfolio selection algorithms can be divided into five classifications, namely, Benchmarks, Follow-the-Winner, Follow-the-Loser, Pattern-Matching, and Meta-Learning Algorithms. The following paper and book gives a good summary of the different strategies and their historical performance. In this notebook, no meta-learners have been implemented. Meta-learners are interesting in that they define several base experts that are combined through some mechanism to form the final portfolio. It might be fruitful to pay more attention to this in the future.

Resources:

Code and Data

HRP

Concept Credit: Marco Lopez de Prado

The problem as stated by Marco is that the Mean-Variance (MV) portfolios are optimal in-sample (training set), but perform badly out of sample (test set). One way to deal with it is to drop forecasts, like risk parity (RP) but both RP and MV require the inversion of a positive-definite covariance matrix. A new method is therefore suggested to overcome the issue of matrix inversion and forecasting, called hierarchical risk parity (HRP). The HRP works by grouping similar investments into clusters based on a distance metric; the covariance matrix's rows and columns also are reorganised so that the largest values lie along the diagonal; lastly, the allocations are split through recursive bisections of the reordered covariance matrix.

You start by defining a distance measure for investments between zero and one $d_{i,j}$ after which you cluster the pair of columns (i^*, j^*) together such that $(i^*, j^*) = argmin_{(i,j)}\{\grave{d}_{i,j}\}\ and\ i\ \neq$ j. Update $\{\grave{d}_{i,j}\}$ with the new cluster and apply steps 3-4 recursively until all N-1 clusters are formed. Then place correlated investments close together and uncorrelated investments further apart. And lastly carry out a top down allocation by assigning unit weight to all items by recursively bisecting a list of items by computing the variance and the split factor and rescaling the allocations, iterate the process until full allocation is achieved.

Resources:

Data, Code

Deep

Concept Credit: J.B. Heaton, N.G Polson, J.H. Witte

A very useful deep learning application for finance are autoencoders. It is a deep learning process to train the architecture to replicate X itself, namely $X = Y$ via a bottleneck structure. An autoencoder can create a more cost effective representation of X. The auto-encoder demonstrates that in deep learning it is not necessary to model the variance-covariance matrix explicitly as the model is already in predictive form. This portfolio optimisation process follows four steps being autoencoding, calibrating, validating and verifying.

Autoencoding: find the *market-cap*, denoted by $F_W^m(X)$, that solves the regularisation problem

$$\min_W \left|\left|X - F_W^m(X)\right|\right|_2^2 \text{ subject to } \left|\left|W\right|\right| \leq L^m$$

For appropriately chosen F_W^m, the process autoencodes X with itself and creates a more information-efficient representation of X.

Calibrating: for a desired target Y, find the *portfolio-map*, denoted by $F_W^p(X)$, that solves the regularisation problem

$$\min_W \left|\left|Y - F_W^P(X)\right|\right|_2^2 \text{ subject to } \left|\left|W\right|\right| \leq L^P$$

This creates a non-linear portfolio from X for the approximation of objective Y.

Validating: find L^m and L^P to suitably balance the trade-off between the two errors

$$\varepsilon_{m=} \left|\left|\hat{X} - F_{W_m}^m(\hat{X})\right|\right|_2^2 \text{ and } \varepsilon_P = \left|\left|\hat{Y} - F_{W_P}^P(\hat{X})\right|\right|_2^2$$

Where W_m and W_p are the solutions to the validation and verification step.

Verifying: choose *market-cap* F^m and *portfolio-map* F^p such that the validation step above is satisfactory.

Resources:

Data, Code, Paper

Linear Regression

We can also apply linear models to the problem of finding optimal portfolios. There is a deep connection between fitting linear models and modern portfolio optimisation. The normal equation is $\hat{\theta} = (X'X)^{-1}X'y$. Note, in statistics of econometrics it is more common to see β $(beta)$ instead of θ $(theta)$. The normal equation makes sense if you do want a linear model and the number of explanatory variables (features) are not too big (e.g., < 100k), perfect for a small universe of stocks. You do not have to use the normal equation, you can also solve for the best coefficients using numerical search such as gradient descent.

To develop this model we regress the excess returns of *N* stocks on a constant =1, without an intercept. Collect the coefficients on each of the stocks in θ. Rescale θ so that the sume of the elements in θ is 1. ($\theta^* = \theta/(1^T\theta)$. The rescaled coefficients are the Markowitz optimal portfolio weights. As a result any ML approach that yields coefficients for a linear model can be reinterpreted as a portfolio optimisation approach; e.g., Ridge regression → Tikhonov regularised optimal portfolio.

In this portfolio-optimisation task, regularisation becomes an important tool to improve out-of-sample performance. These regularisation methods include l2-norm (Ridge Regression), L1-norm (LASSO), combination of L1 and L2 norms (Elastic Net Regression). Regularisation effectively helps you to manage the variance-bias trade-off. Britten-Jones (1999) demonstrates a close equivalence between Markowitz portfolio optimisation and OLS regression.

Resources:

Code, Paper

Further please see this for an article I wish I wrote, for a few wise words on portfolio optimisation.

PCA and Hierarchical

The optimisation method involves integrated portfolio selection and risk estimation steps to optimise the portfolio. Principal Component Analysis (PCA) and clustering techniques are used to build classes of similar assets. Here I will primarily focus on the function of PCA.

Let $Y = (Y1, \dots, Yn)^T$ denote an n-dimensional random vector with variance-covariance matrix, $\sum$. The goal of PCA is to construct linear combinations $Pi = \sum_{j=1}^{n} W_{ij} Yj, for\ i = 1, \dots n$ in such a way that the Pi's are orthogonal so that $E[PiPj] = 0$ for $i \neq j$ and so that the Pi's are ordered to explain the largest percentage of total variability in the system and each Pi explains the largest percentage of

total variability in the system that has not already been explained by $Pi, \dots, Pi - 1$. Once that is done, you can select an optimum level of components and apply this technique to a universe of stocks.

Resource:

Code

Risk

GANVaR

Credit: Hamaad Shah

By using GAN (specifically BiGAN) on financial data, we do not have to explicitly specify the distribution (e.g., multidimensional Gaussian distribution). This model can be used to estimate portfolio risk measures like the Value-at-Risk (VaR). GAN allows us to maximise the likelihood of complex distributions, this might be key for financial data know to be complex and highly non-linear. This method is compared against traditional method. The math is presented inside the notebook.

Resources:

Code

Execution

Many sell-side firms have experimented with algorithms that use reinforcement learning for execution. These algorithms typically consist of a scheduler that makes macro decisions across a time horizon of hours, and a limit order module that makes micro decisions on smaller time scales. The limit order modules is based on a reinforcement learning model. First, identify your current state or regime and then take micro actions. Within a stated horizon, the limit order module seeks to sell and buy securities with for example the aim of minimising slippage.

A reinforcement model needs to have appropriate data. Current implementations include market size/price data (e.g. spreads), execution data (e.g. filled quantity and realized p.o.v. rate), market signals (e.g. medium-frequency like momentum and high-frequency like order flow) and model estimates (e.g. volume/volatility prediction and fill probability estimate). In addition, as mentioned before, the model operates under the constraints of the quantity, time horizon, limit price and other market conditions. The reinforcement model decides to either place an aggressive cross the spread or passive order at each price level of the order book. An inherent problem is that rewards attributable to a step can only be discerned at the end of all transaction, hence the need for reinforcement learning. In most examples, good success can be achieved by learning the functional map via a neural network with one hidden layer. The point is to learn the value function within the framework of reinforcement learning.

Validation Techniques and Data Processing

ML Conceptual Map

Machine learning techniques can largely be broken into the data processing, supervised learning, validation techniques, unsupervised learning and reinforcement learning. Every machine learning solution is constructed out of a mixture of these components. In the short-term, I see a lot of scope for innovation w.r.t. unstructured data processing techniques and in the long run a lot of room for

improvement in the reinforcement learning space. In this section, I will look at some of these components through an asset management lens by pulling some additional information from the previous trading strategies.

1. **Data Processing**
    a. Natural Language Processing
        i. Text Extraction
        ii. Word Embeddings
        iii. Topic Modelling
    b. Image and Voice Recognition
    c. Feature Generation
2. **Supervised Learning**
    a. Algorithms
        i. Gradient Boosting
            1. LightGBM
            2. XGBoost
            3. Constraint
        ii. Neural Networks
            1. CNN
            2. RNN
            3. GAN
    b. Tasks
        i. Regression
        ii. Classification
    c. Analysis
        i. Cross Sectional
        ii. Time Series
3. **Validation Techniques**
    a. Visual Exploration
    b. Table Exploration
    c. Feature Importance
    d. Feature Selection
    e. Cross Validation
4. **Unsupervised Learning**
    a. Traditional
        i. Dimensionality Reduction
        ii. Clustering
        iii. Anomaly Detection
        iv. Group Segmentation
    b. Neural and Deep Learning
        i. Autoencoders
        ii. Boltzmann Machines
        iii. Deep Belief Networks
        iv. Generative Adversarial Networks
    c. Semisupervised Learning
        i. Mixture Models and EM
        ii. Co-Training
        iii. Graph Based
        iv. Humans
5. **Reinforcement Learning**
    a. Markov Decision Process and Dynamic Programming
    b. Monte Carlo Methods
    c. Temporal Difference Learning
    d. Multi-armed bandit
    e. Deep (Recurrent) Q Network
    f. Actor Critic Network
    g. Inverse Reinforcement Learning

Data Processing:

Feature Cleaning

There are a few cleaning operations you want to do before starting a prediction task. This includes dropping columns that are mostly empty, dropping constant columns, dropping quasi-constant columns, dropping columns that are overly correlated (only in prediction task, not in predictor analysis tasks), dropping rows that are mostly empty across columns, and identifying and replacing outliers (if needed). For linear and neural network models, it is often the best strategies to clip outliers rather than dropping the samples.

You can fill empty values with column median or reconstructed values. I would normally set the missing threshold to 0.9 and the correlation threshold to 0.99. I would only drop correlated features to improve prediction performance; this should not be done when you want to perform a predictor analyses as you risk removing extremely important features from the analysis. With GBM models, you can also fill null values with out of range values (-999, -1 etc.), only do this after feature transformations, otherwise these strange values might be caught up in mapping operations and on a further note do not do this for neural networks. Before you remove null values, you can also create an extra binary column to define whether a null value has appeared, making sure that we do not lose information if the column was not missing at random. In time series always do back filling instead of forward filling when working with features, and although I recommend dropping

Resources:

Code

Feature Generation

Signal Processing

Financial data is fundamentally noisy, making data processing one of the most crucial steps. It cannot be neglected and is essential for all learning problems. The data processing starts at the point of data selection. Something as simple as knowing to select ETF's as opposed to reconstructed indexes can significantly improve the generalised accuracy of your model as a result of removing survival bias.

Similar to statistics, it's best to work with relatively invariant processes such as the returns on price, and the changes in yield and volatility. The mean variance and auto-correlation should be near constant over time. Researchers like Charles Tapiero (NYU) have looked into fractional finance, which consists of fractional derivatives of price series. Derivatives on the price of securities can be optimised to balance the need of stationary series with the need of predictability.

Marcos Lopez de Prado have phrased this question differently as "what is the minimum amount of differentiation that makes a price series stationary while preserving as much memory as possible". To me this can only be resolved in one way, and that is by experiment, I personally do not see the need for stationarity to persist when a level of non-stationarity works in your favour. To make data 'more' stationary, one can therefore look at differentiation and fractional differentiation methods. To generate more fundamental and market data, one can make use of GAN while considering the dollar additivity constraint. I will release some notebooks touching on these subjects in the future.

To deal with some of these issues one can decompose a time-series into the trend, seasonal and residual components. These components can be separately included into a machine learning model. An ARIMA model itself can be included as an input to a model. For example, in time series classification, we need not stop with the ARIMA model, I have successfully included instance based

models like Dynamic time warping (DTW), Weighted DTW, Time Warp Edit Distance (TWED), Moving-Split Merge (MPM) and derivatives DTW (DDTW) in my meta-model.

Automated Engineering

It is good practice to make use of automated feature engineering tools. “Having too many features is a better problem to have than having too few” – some wise sage. The automated tools that can be particularly helpful includes, TSFresh, Feature Tools, SHAP features and some others. GBM model specifically struggles to approximate multiplication and division, therefore adding these transformations explicitly can improve your model.

Manual Engineering

Categorical Encoding

When a feature has high cardinality and is sparse, it is good practise to test the performance of the feature before clogging up your model. Another important consequence of including any feature is the knowledge of whether or not you would have access to the feature in the future. If you do not, it is unwise to include it in the present model. Also consider a cost-benefit of features, when features are too costly to obtain and do not provide much of a benefit, do not include it. Data acquisition has to be value driven.

For neural network and linear models, when your dummies explode you can use sparse matrixes, which works by storing only non-zero elements to save memory. Most GBM models can also work with sparse matrixes directly. A good feature for neural networks is the interaction between categorical variables; this is not useful for GBMs. You can do numerous other transformations like Ranking, Quantization, Binning, Transformation and Mapping, Clustering and Data-time extraction.

Preprocessing

In the Preprocessing step, you have to normalise and or standardise your data. You also have to decide whether your target variable is normally distributed. If it is not, you can do a log transform or a square root transformation. Sometimes it is helpful to train a model on different preprocessing groupings and selecting the best among them. Data transformations including unsupervised feature creation should be separately done on train and test sets to avoid cross-contamination. Researchers often create a ‘pipeline’ for this purpose to easily create the transformation for each set.

Resources:

Code

Feature Selection

All machine learning models internally perform a sort of variable selection in the training process. LASSO regressions do this quite explicitly by zeroing out coefficients. Gradient Boosting Models (GBMs) will on the other hand ignore non-informative features for the splitting operation, as shown by the low importance score these features obtain. For GBMs, we can measure this importance measure because of the use of variables to split nodes. Each split will give us a measure of expected improvement, i.e., decease in the impurity of the node; and we can record these improvements over

all nodes and all trees. I personally do not give much time to any of these measures, as they are mostly misleading.

I do not perform any feature selection beyond the feature cleaning stage for prediction tasks; for predictor analysis tasks, I do however perform extensive feature selection. When I do feature selection for prediction tasks, it takes the following form. Using out of sample data, I would calculate the feature importance using permutation importance and SHAP values. I would then remove values so that the cumulative importance is 99% for both of these methods, which normally drops about 10% of the original features.

In the age of deep learning, I would strongly advise against *other* feature selection methods. They are poor proxies of performance once ingested into a model. Many teams, including Google, use large models with a lot of data without bowing down to feature selection. In my personal experience with GBMs and NNs feature selection has become mostly redundant. In saying that sometimes feature selection has economic reasons behind it, e.g. which features can I remove while still maintaining 90% accuracy so that I can stop paying the expensive data provider? In addition, some features simply introduce noise that can hurt out of sample performance, in which case you can remove feature until you hit 99% of cumulative feature importance as described in the before mentioned paragraph.

Validation

Exploration

Table Exploration, visual exploration and two.

Model Selection

Table 3: XGBoost and Deep Learning Model Performance Comparison

Metrics	XGBoost Model	Deep Feed Forward Network	Deep Convolutional Neural Network	Logit Model
ROC AUC Sore	0.9587	0.8444***	0.9142***	0.7092***
Accuracy Score	0.9755	0.9324	0.9518	0.6856
False Positive Rate	0.0037	0.0666	0.0296	0.2296
Average Log Likelihood	0.1414	0.5809	0.2996	1.1813

This table illustrates the performance of two deep learning models against the XGBoost Model. The Feed Forward Network is a deep learning network that does not circle back to previous layers. The Convolutional Neural Network is a biologically inspired variant of MLP, popularised by recent image classification studies. The best possible Logit model was established by choosing a selection of the best variables. Further results include the isolation of the 10 best predictor variables (using the Gini Index) in all models, this produced similar results to the above table both in extent and in rank. *p<.1 ** p<.05 *** p<.01. Significance levels are based on a two-tailed Z test to identify the statistically significant difference between all contender models and the best performing model, which is made possible due to the cross-validation process.

XGBoost Model Comparison Using Different Inputs

Metrics	All Data	50 Variables Model	One Year Before Bankruptcy	Two Years Before Bankruptcy
ROC AUC Sore	0.9587	0.9408***	0.9666**	0.9434***
Accuracy Score	0.9755	0.9700	0.9860	0.9837
False Positive Rate	0.0037	0.0056	0.0010	0.0002
Average Log Likelihood	0.1414	0.1795	0.1682	0.2206

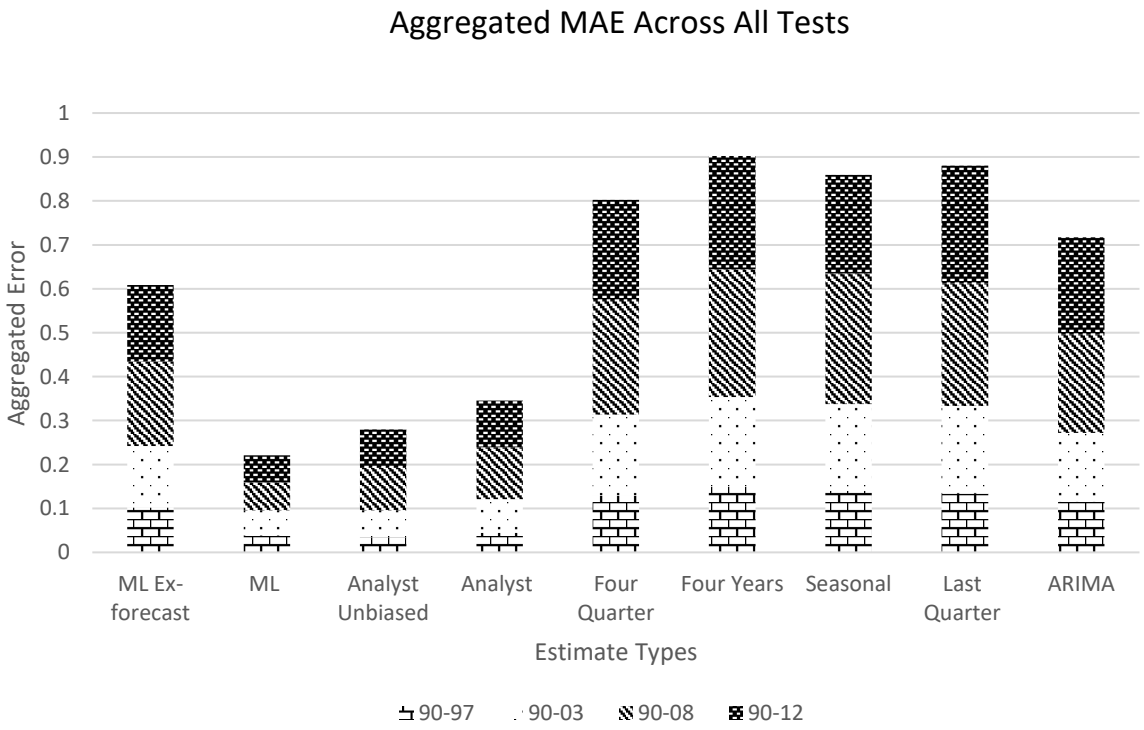
This table compares the performance of model that includes only 50 of the most predictive variables as inputs, a model that only includes bankruptcy observations one or two years before the filing. All statistical comparisons are made against the model called "All Data." *p<.1 ** p<.05 *** p<.01. Significance levels are based on a two-tailed Z test.

Metric Validation

For time series validation, I generally benchmark against the lower of (Theta+ARIMA +ETS)/3 or ARIMA. For many examples you would struggle to beat this benchmark. Some more time-series notebooks can be found here (rough). The ARIMA model can also be automated.

Regression Example (Earnings Prediction):

From left to right, the above models relate to the following, as presented in the equations in the study, $\hat{p}, est\_avg_t, EpsEst4Q, EpsEst4Y_{it}, EpsEstRW_{it}, EpsEstLQ_{it}$ and $ARIMA$.



This figure reports the aggregated MAE tests over multiple test periods. The four periods and associated patterns at the bottom of the graph. The above chart presents both an ML model that does incorporate analysts' forecasts (ML), and a model that does not incorporate analyst forecasts (ML Ex-forecast) It is, therefore, unlike the time-series models that only use past actual EPS values. Further, using an OLS regression the Analyst Forecast has been bias corrected (Analyst Unbiased).

Classification Example:

For classification, you can use a random confusion matrix. The evaluation of a successful model can start with an accuracy measure. The accuracy can be defined as the percentage of correctly classified instances (observations) by the model. It is the number of correctly predicted surprises (true positives) and correctly predicted non-surprises (true negatives) in proportion to all predicted values. It incorporates all the classes into its measure $(TP + TN)/(TP + TN + FP + FN)$, where *TP, FN, FP* and *TN* is the respective true positives, false negatives, false positives and true negatives values for all classes. The measure can otherwise be represented as follows: $acc(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i = y_i)$.

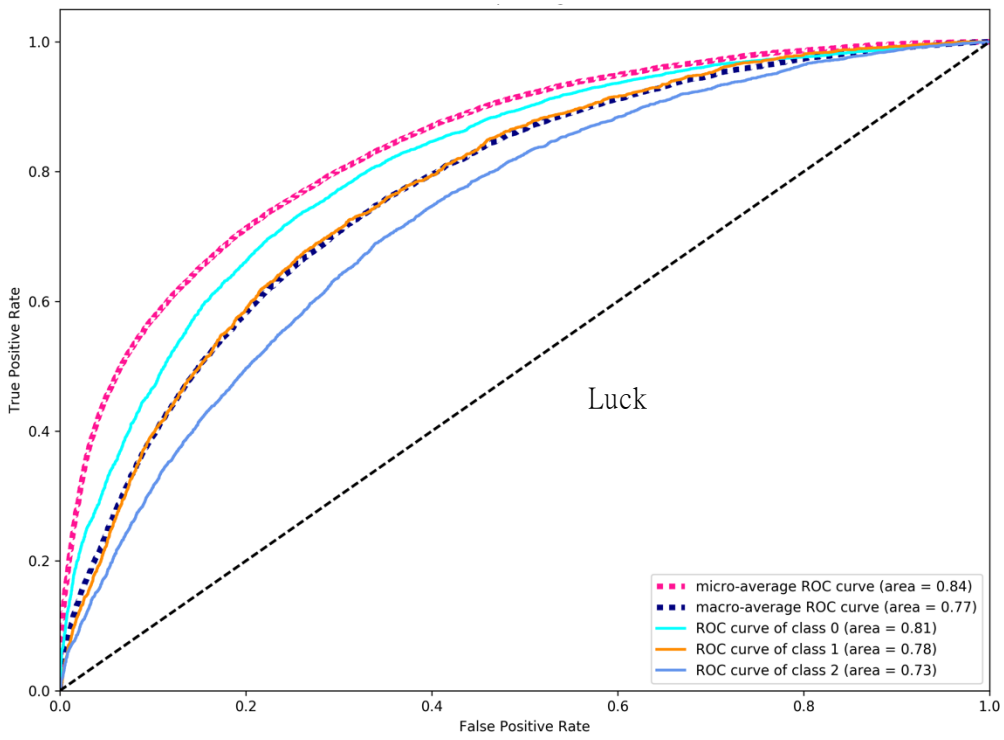
Surprise Breakdown Random Guessing Confusion Matrix

Random Confusion Matrix		**Random Guessing**			Marginal Sum of Actual Values
		Neutral	Negative	Positive	
Actual	Neutral	**89020**	24590	55890	169500
	Negative	24590	**6792**	15439	46821
	Positive	55890	15439	**35090**	106419
Marginal Sum of Predictions		169500	46821	106419	322740

This table is formed by 'randomly choosing the observations' by allocating the observations according to the underlying distribution.

The multiclass ROC is a universal way to identify the performance of a classification model. The AUC (area under curve) score provides an integral based performance measure of the quality of the classifier. It is arguably the best single number machine learning researchers have in measuring the performance of a classifier. The middle line is a line of random ordering. Therefore, the tighter the ROC-curves fit to the left corner of the plot, the better the performance. Two other measures included in the graph is a macro-average measure that equal weight to each class (category) and a micro-average measure that looks at each observation. AUC values of 0.70 + is generally expected to show strong effects. The ROC test is the first indication that the model performance is significantly different from null. In subsequent tables.

Multiclass Receiver Operating Characteristic (ROC) for a 15%+ Surprises Strategy



This figure reports the ROC and the associated area under the curve (AUC). The ROC is measured for three different classes, class 0 is the negative surprise class, 1 is the neutral class and 2 is the positive surprise class. The macro-average measure is equal weighted to each class and a micro-average measure looks at each observation weight. The random ordering or luck line is plotted diagonally through the chart. The associated curves are associated with a good classification model.

Healthy and Bankrupt Confusion Matrix.

Aggregated Health and Bankrupt Firms Matrix		**Predicted**		Sample Proportion
		Healthy	Bankrupt	
Actual	Healthy	**29041 - TN**	116 - FP	0.96
	Bankrupt	805 - FN	**258 - TP**	0.03
Precision		0.97	0.69	30220
Improvement		0.01	0.66	-

This bankruptcy prediction task solves for a binary classification problem that produces a 2×2 matrix. The columns of the matrix represent the predicted values, and the rows represent the actual values for bankrupt and healthy firm predictions. In the cross-section of the rows and columns, we have the True Positive (TP), False Negative (FN - type II error), False Positive (FP - type I error)), and True Negative (TN) values. The sample proportion on the far right is equal to all the actual observations of a certain classification divided by all the

observations. The *precision* is calculated by dividing the true positives (Bankruptcies) with the sum of itself and the false negatives (Healthy). An example along the second column: $258/(116 + 258) = 69\%$. The improvement is the percentage point improvement the prediction model has over a random choice benchmark.

Random Guessing Confusion Matrix.

Aggregated Health and Bankrupt Firms Matrix		**Random Guess**		Marginal Sum of Actual Values
		Healthy	Bankrupt	
Actual	Healthy	**28131 - TN**	1026 - FP	29157
	Bankrupt	1026 - FN	**37 - TP**	1063
Marginal Sum of Guesses		29157	1063	1063

This table is formed by 'randomly choosing the observations' by allocating the observations according to the underlying distribution, as presented by Sample Proportion in **Error! Reference source not found.**.

Cross Validation

There is no reason why you cannot test which validation process leads to the best out-of-sample performance. Sometimes you can be surprised by the results. Rolling, similar to TSF, is not commonly used in ML literature, but it perfect for time series prediction. It always outperforms train-test division unless randomness intervenes.

Table 4: Model Comparison Using Different Performance Validation Procedures

Metrics	(1) All Data	(2) Time-Split (TS)	(3) K-Fold (KF)	(4) Time Split Fold (TSF)	95% Confidence (+/-)
ROC AUC Sore	0.9587	0.9655**	0.9467**	0.9570	0.0142
Accuracy Score	0.9755	0.9837	0.9682	0.9712	0.0163
False Positive Rate (p-value)	0.0037	0.0069	0.0028	0.0039	0.0015
Average Log Likelihood	0.1414	0.0825	0.1301	0.1052	0.0707

This table compares the performance of the best models that resulted from different out-of-sample performance tests. (1) The original "All Data" model allocates 60% of the observation to the training set, 15% to the development of validation test set and 25% to the test set. The 15% is used to measure and improve the performance of the model. The observations to each of the splits is randomly selected. (2) TS is a simple ordering of the observation in time series and the creation of longitudinal training - 60%, validation - 15% and test set splits -25%, this method ensures that there is no information leakage from the future observations. (3) KF is a randomised cross-sectional method that scrambles the observations and splits them into training and test sets and calculates the average metrics from 10 different iterations or folds. (4) TSF is the most robust method and has also led to the model with the best generalisable performance as evidenced by the battery of metrics - It is a longitudinal blocked form of performance-validation that suits this form of bankruptcy prediction, it uses the

strengths of both (2) and (3). All statistical comparisons are made against the model called "All Data." *p<.1 ** p<.05 *** p<.01. Significance levels are based on a two-tailed Z test.

Feature Value

The following models have model-specific feature interpretability: linear regressions, logistic regressions, GLMs, GAMs, Decision Trees, Decision Rules, RuleFit, NBCs and KNNs. For more black-box models one can use model agnostic methods like partial dependence plots (PDPs), individual Conditional Expectation (ICEs), Accumulated Local Effects (ALEs), Feature Interactions, Global and Local Surrogates, and lastly Feature Importance measures.

Feature importance measures can be model-specific or model-agnostic. The only methods that I trust, albeit cautiously, is Permutation Importance and SHAP Values. Both of these methods are model-agnostic. I also recommend SHAP dependence plots as opposed to PDPs. However, PDPs and ICEs can be successfully used to identify feature and feature interaction behaviour when you are using GBMs and a gain measure. So start with SHAP values and Dependence plots and for additional values move into GBM's with PDPs + ICEs.

Per the above, the only feature *importance* measures worth studying are permutation methods. Permutation is the random shuffling of feature values to nullify a feature, which allows you to investigate how much worse a prediction model is performing without the feature. The first is a method known as Permutation Importance (PI) and the second SHAP values (SVs). For both of these methods, you can estimate significance scores. Compare to normal Shapley values, I prefer SHAP values because it returns sparse results with many of the shapley values estimated to be zero; this is especially useful in feature selection procedures.

One of the disadvantages with SV is the compute which is O(2^n) order of magnitude. Generally, one wants to avoid algorithms with running times where n is an exponent. The SV can be interpreted as the contribution of a feature value to the difference between the actual prediction and the mean prediction. So, what is PI. It can be defined as the increase in the prediction error of the model after the feature's values are permuted. Features' PIs changes with each random shuffling. It is therefore good practice to repeat the permutations and develop a statistical significance score. Both of these values can be used in regression and classification tasks, for regression one would typically look at accuracy score and for classification, one would typically look at the ROC (AUC) score.

The benefit of SVs is that they provide local and global insight but this is at the expense of being slightly less interpretable. The disadvantage of PI is that it only provides global insight but on the other side are very factual and intuitive. These measures play complimentary roles in all my analyses. When you are investigating global feature importance look at PI, when you look at local feature importance, or and interested in interaction effects, individualised data points then use SVs. For feature selection only use training data, for predictor analysis and feature importance use both training and test data. Just a last word of warning, when features are correlated then the overall scores would be much lower as the remaining features pick up the slack. This is true for all permutations methods. I have devised a method to overcome this problem, called *interpretive-features*.

When we collapse a feature's importance across all samples to a single number we are forced to decide what we want to measure. For example, is a feature with high effect on a small number of observations more important than a feature with a small effect on many observations? I think the answer to this depends on the application. With SHAP values you are able to investigate how the feature 'acts', this cannot be done with Permutation importance. To understand what is meant with SHAP values being local and individualised, consider the following. After you train your model and explain it on your training dataset to get a matrix of SHAP values, you can consider every single column as the importance of a feature across all samples. It will have both positive and negative values.

One can do many extended statistical tests by having access to this data. For one, you might want to know if those values have a meaningful trend or are just driven by random noise. To evaluate this you can retrain your model on a bootstrap resample of your dataset and then explain it again on your original training data to get another matrix of SHAP values. If you take the dot product (or correlation) between two of the same columns in each matrix you will see how well the impacts of a feature in the first model agree with the impacts of that same feature in the other model. By repeating this multiple times you will get an estimate of the global stability of a feature. If the correlation is consistently greater than 0 then you have a stable feature. With additional bootstraps you can also build a global-level importance interval for each feature and calculate the associate feature significance.

A further neat attribute of Shapley values are their linearly additivity. As a result you can compare groups together by aggregating the scores. You might wonder how SHAP works, it works by permuting input data to assess the impact of each feature. Each feature's contribution is averaged across all possible feature interactions. This approach is based on the concept from game theory. Permutation importance does something similar, but only assess the global impact on the prediction, it is a traditional feature importance algorithms. All traditional feature importance algorithms will tell us which features are most important across the entire population, but this is one-size-fits-all approach that does not always apply to each individual *sample* (company, customer, transaction, and unit). A factor that is an important driver for one *company* may be a non-factor for another. By looking only at the global trends, these individual variations can get lost, with only the most common denominators remaining.

With individual-level SHAP values, we can pinpoint which factors are most impactful for each individual *company*, allowing us to customize our next actions accordingly. When features are correlated, their impact on the model score can be split among them in an infinite number of ways. This means that the SHAP values will be lower than if all but one of the correlated feature(s) had been removed from the model. The risk making some features look less important than if their impacts remained undivided. To be fair, all known feature importance methods have this problem.

Interpretive-Features

Interpretive-features is a recursive numerical approach to identify the most proximate features to the target value conditioned on a neural network model. The problem is that SHAP and Importance Permutation assumes independent features. The strength of this method is that it does not assume feature independence. It identifies the lowest model-redundant, highest performing feature set through a neural network approximator. This is the best method I know of to deal with multicollinearity when performing a predictor analysis as opposed to prediction task. This method maximises the interpretability and explainability of a model mathematically. Note, although possible, this method is not meant to improve your prediction accuracy, and instead is meant to highlight the best features to describe the movement of the target variable.

For *n* respective features select their closest five correlates and add 5000 additional bootstrapped rows by randomly sampling from the five top correlates; collectively call them *k*; establish model *m(n_k);* cycle through all possible model combinations; train the model on a correlation aggregated time series and test the model on individual correlates. Select the set of remaining features of the top performing model as the low-correlation-high-prediction model. Repeat the process if necessary. This is a model-specific technique to get rid of unnecessary correlated features. This concept is built on the theory that the correlates with the highest predictive power are causally more proximate to the target or response variable.

For more detail, see the steps as outlined below.

1. Identify each feature's (F) top *N* (5) correlates.
2. Normalise all features.
3. Combine and average all correlates into one time series.
4. Train a multi-layered neural network (*M*) with the newly horizontally aggregated time-series.
5. Resize top *N* correlates table to number of simulations requested (S= 5000)
6. Permute the table of correlates to obtain 5000 independent simulations
7. Test each of the 5000 independent feature combinations on the trained model *M.*
8. Select the top decile performing simulations.
9. Get the mode feature of each decile (500 S) for each Feature placement holder (F).
10. Remove duplicates (Some features selected multiple times, because they outperform correlates).
11. With smaller group of features start again with the permutation step at (5).
12. Iterate for three different neural network node constructions and three different seeds.
13. Find the mode features for the final iteration; generally about $1/5^{th}$ of original features.
14. Retrain a model on the mode of the last iterations (i.e. best lowest-correlative highest-performing model)
15. Construct a special permutation importance method that removes multi-collinear issues.
16. Iterate by retraining model in (14) on five different seeds to obtain additional permutation importances.
17. Calculate the doubly correlation adjusted p-values, select feature where p-value > 0.05 (at this point should be about 90%+ of all features)
18. The result of the p-value table is the result of the features that are the most proximate to the target value.
19. At this point you can do a SHAP analysis to further highlight feature characteristics.

In my notebook, I have hardcoded some values, but this model is flexible and these parameters can be automated. Although this method is experimental, it has shown to produce good results (It can actually be tested with causal analysis). Also, instead of the feature-specific approach above, you can also grab the first PCA component from a group of correlated features. You can get to understand how these features behave collectively and what characteristics they exhibit. You can then use this as new features to the model. I rarely do this, as I am interested in the best performing features as opposed to the best performing group of features.

Another approach would be to permute all correlated features together and assign to them a mutual Permutation importance. When you use GBMs you can additionally use column-wise subsampling during model training in the presence of correlated features. This is roughly similar to Ridge regression for linear models in that it tends to evenly spread out importance across correlated features.

Predictor Analysis Example:

Examples from earnings, bankruptcy and restaurant prediction.

Earnings Related **Variable Importance** and Response Direction for Classification

Name	Short Description	Score	D
est_avg_t	This time period's analyst EPS forecast	0.247	-
$diff_{-1}$	The difference between the past actual EPS, p_{-1} and p_{-2}	0.119	-/+
p_{-1}	Actual EPS t_{-1}	0.082	-
$d\_e\_diff_{-4}$	Difference between the past actual, p_{-4} and forecast est_avg_{t-4}	0.073	+
$diff_{-4}$	The difference between actual EPS p_{-4} and actual p_{-1}	0.060	-/+
Other	57 other earnings-related variables.	0.212	
Total		**0.794**	

This table identifies the most important variables as identified by the Gini importance which measures the average gain in information. The variable importance measure (Score) is based on all variables. *D* identifies the direction of the 'coefficient' as identified by the partial dependence process.

The most important earnings-related variable is the forecast itself, est_avg_t this is expected because the purpose of the model is to identify deviations from this amount and the actual, it, therefore, provides a measure of reference to the other variables of the model. Countless papers have identified the performance of analysts' forecasts in forecasting earnings as recapped by Brown (1987b). The lower the forecasted EPS the more likely a surprise is to occur all else equal, 32% of the outcome is attributable to this value. A lower EPS could proxy for a smaller firm or any other attributes. The second and fifth most important variable, is the difference between the actual earnings between t_{-1} and t_{-2}, called $diff_{-1}$, and the difference between t_{-1} and t_{-4}, called $diff_{-4}$ These are novel variables not yet identified by past research. It basically says that the past increases in earnings are an important variable for predicting future surprises, which makes intuitive sense. If the value is very high, surprises become more likely. Surprises are also more likely if the value gets very low. For a lot of the observations, the value is very small, decreasing the likelihood of surprises. The measure is u-shaped, which is indicative of a sort of variance measure. The next important value is the actuals earnings at time t_{-1}, called p_{-1}.

Research by Bradshaw et al. (2012), have shown that the past annual earnings, often outperform, not just mechanical time series models, but also analyst' forecasts. Similarly, past quarterly earnings also seem to be an important value in predicting the next quarter's earnings surprise and is similarly shaped to the analyst forecast, est_avg_t. The relationship shows that where p_{-1} is large and est_avg_t is simultaneously low, then a positive surprise is likely to occur more than 90% of the time, all else equal. Further, where p_{-1} is low and est_avg_t is high then a surprise is unlikely to occur. The next important variable is the difference four quarters ago, i.e., one year between the forecast, est_avg_{t-4} and the actual value, p_{-4}. The importance of this variable was also expected as Easterwood & Nutt (1999) and Fried & Givoly (1982) separately showed that past errors tend to persist. The larger the difference, the higher the likelihood of surprise. Other variables that showed an above 2% importance includes rolling

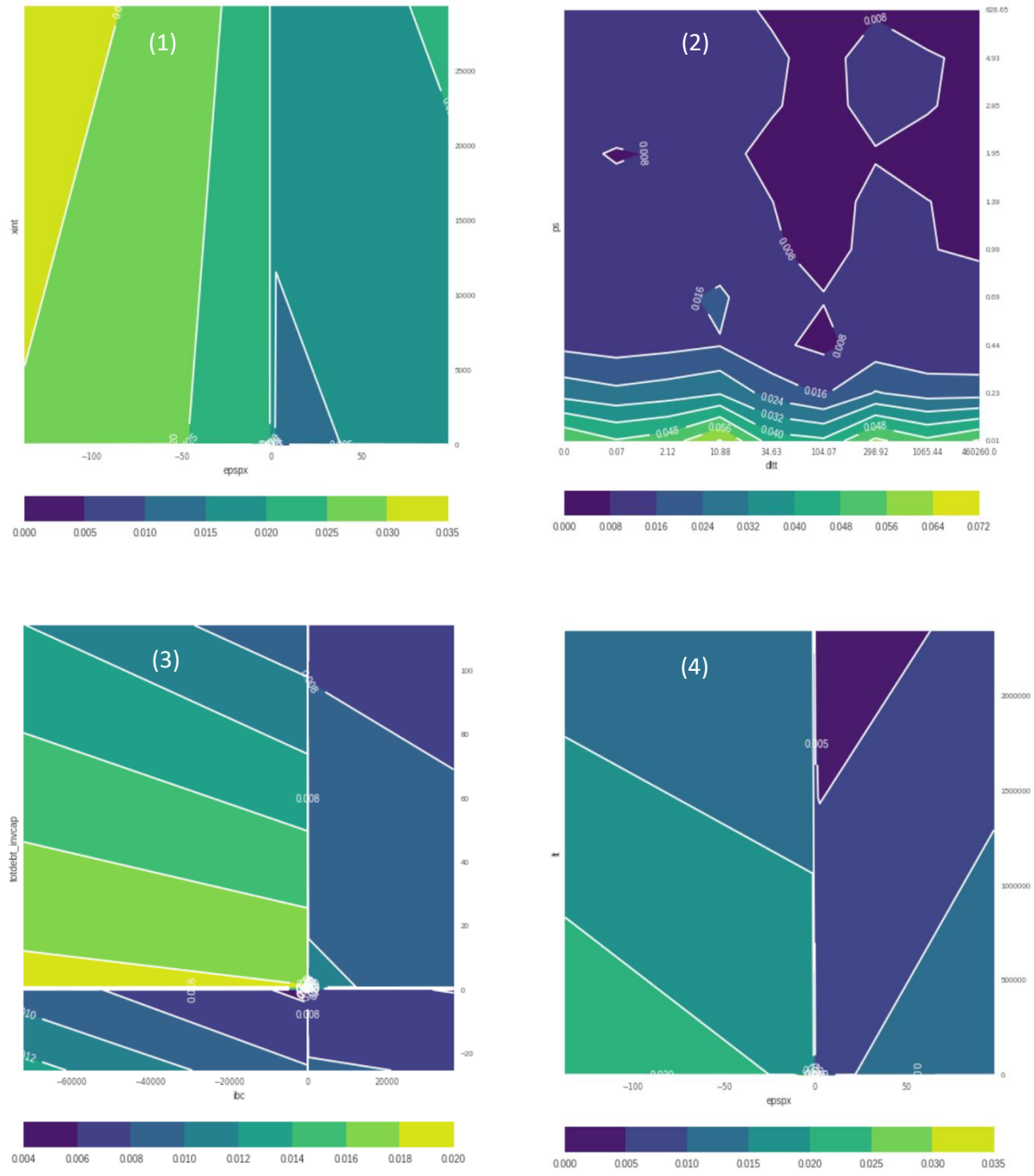
averages and weighted rolling averages of the difference between past earnings and analyst forecasts, and the standard deviation of analyst forecasts.

Category Importance Analysis (Multiple **Importance Measures**)

Category	(1) CI	(2) RCLI Top 50	(3) Post GFC RCLI	(4) Pot	(5) F	(6) wF	(7) 24 CI	(8) PCA 10	(9) RIT	(10) Avg.	(11) Fin.
Assets & Liabilities	1	1	1	1	1	1	1	1	1	1.0	1
Solvency	2	3	2	2	2	2	3	2	2	2.2	2
Income	3	2	3	3	4	5	2	4	3	3.2	3
Valuation & Profitability	4	4	4	4	3	4	5	3	4	3.9	4
Equity	5	5	6	5	5	3	4	6	5	4.9	5
Expense	6	6	5	6	7	6	6	10	7	6.6	6
Efficiency	7	7	7	8	6	7	7	9	6	7.1	7
Other	8	8	8	9	8	8	10	8	8	8.3	8
Liquidity	9	9	9	10	8	9	9	5	10	8.7	9
Cash Flow	10	10	10	7	10	10	8	7	9	9.0	10

This table is an attempt to regroup categories where there is a strong correlation 80% + and to calculate the rank of the categories according to 9 different predictive importance strategies. This table calculates the equal weighted average of nine ranking methods (10). (1) Is the normal importance measure (gain measure) calculated for all variables in every category. (2) Is the gain measure for only the top 50 variables. (3) Is the gain measure after the GFC. (4) Is the ranking according to the potency measure, being the category importance weighted by the number of variables in each category. (5) Is a measure (FScore) that tallies the amount of variable splits across all trees within each category. (6) Measures the FScore weighted by the probability of the splits to take place. (7) Is the overall gain measure for a newly created model that only incorporates the 24 best variables. (8) Is the importance of the first PCA component for each category. (9) Avg. is the equal-weighted rank average for each category. (10) Is the final ranking obtained by ranking the raw average. When percentage growth measures were removed from the categories, all categories remained unchanged apart from a swap between *Other* and *Liquidity*. A further split in category where solvency ratios were split between capital structure, and coverage and cash flow ratioratios resulted in the following rank among categories, (1) asset and liabilities (2) Income (3) *valuation and profitability*, (4) *capital structure,* (5) equity, (6) *interest coverage,* (7) expense, (8) *efficiency*, (9) *cash flow* ratios, (10) other ratios (11) *liquidity ratios*, (12) cash flow values. The ratio values are italicised.

Interaction Pair **Partial Dependence Plots (Depth Two)**



(1) Top left is the interaction between Interest and Related Expense (xint) and the EPS Excluding Extra. Items (epspx) and resulting response. (2) Top right is the interaction between Price to Sales (ps) and Long-Term Debt (dltt). (3) Bottom left is the interaction between Total Debt to Invested Capital (totdebt_invcap) and Income Before Extraordinary Items (ibc) and the interaction effect on the bankruptcy outcome. (4) Bottom right is the interaction between Total Liabilities (lt) and the EPS Excluding Extra. Items (epspx).

Cross Tab - Top **Variable Interactions**

	totdebt_invcap	ps	lt	xint	pi
ibc	779	704	63	45	13
pi	66	585	209	338	0
epspx	228	76	551	509	34
dltt	17	418	156	34	14
debt_assets	43	127	239	77	390
ppent	71	279	82	28	61

This table represents the most important interaction pairs as measured by the gain statistic at an interaction depth of two. The table has been constructed to highlight the top ten interactions. For completeness, the surrounding interactions have also been included. Variables vertically follows, Income Before Extraordinary Items (ibc), Pre-tax Income (pi), EPS Excluding Extra. Items (epspx), Long Term Debt (dltt), Total Debt to Total Assets (debt_assets), Property Plant & Equipment (ppent). And horizontally, Total Debt to Invested Capital (totdebt_invcap), Price to Sales (ps), Total Liabilities (lt), Interest and Related Expense (xint).

Interaction Analysis – Depth Three

	Term 1	Sign	Term 2	Sign	Term 3	Sign	RII	Gain
(1)	epsfx	-	ibc	-	totdebt_invcap	+	100	456
(2)	debt_at	+	pi	-	rd_sale	-	95	435
(3)	dpc	-	equity_invcap	+	ps	-	92	419
(4)	ibc	-	ps	+	totdebt_invcap	+	88	402
(5)	dltt	+	ibc	-	ps	-	84	383
(6)	dltt	+	pi	-	ps	-	84	382
(7)	ibc	-	ps	-	ps_prtc	-	83	378
(8)	ibc	-	ibc	-	totdebt_invcap	+	79	362
(9)	dltt	+	ps	-	txt	+	76	348
(10)	ibc	-	ppent	+	ps	-	74	336
(11)	at	+	debt_at	+	epspx	-	68	310

Out of the top 50 variable list, there are millions of ways to conjure up directional relationships. Due to the nature of nonlinear relationships, to conceptually understand the web of relationship, it is best to identify the top interaction pairs. This table represents the most important interaction pairs as measured by the gain statistic at an interaction depth of three. For easier reading, I also report the relative interaction importance (RII). The sign purely indicates the average direction of each variable. The interaction terms are much more informative than single standing variables. Interactions is at the core of what gradient boosting tree models are all about. Unique *Terms 1* are EPS (Diluted) - Excl. Extra. Items (epsfx), Assets - Total (at). Unique *Terms 2*

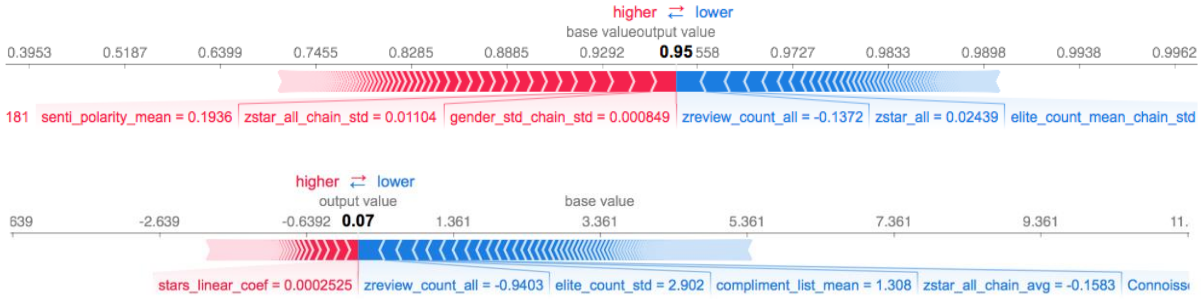
are Common Equity/Invested Capital (equity_invcap). Unique *Terms 3* are Research and Development/Sales (rd_sale) and Income Taxes - Total (txt).

Bubble Plot and Ranking of each Model's Most **Important Categories.**

This figure reports the relative importance of the five outcome classification models and the associated accounting dimensions. There is a large amount of heterogeneity between the different classification models.

An important benefit of SHAP is that it provides the aggregate direction a feature has in relation to the response variable. *Figure 2* shows how the SHAP values interact with each other to produce the final output. Although, a few variables cause a large amount of the movement, *Figure 2* also shows that less important variables play a large role in aggregate due to the sheer number of variables present. *Figure 3* shows these predictions vertically while sorting them by similarity to show the effect these different features have on the final outcome. Here, I have only plotted a subsample of a hundred random observations. The final output is where the red and blue stacks meet. Each vertical slice is a DNA strand of sorts displaying the characteristics driving the predicted outcome of each observation.

Figure 2: Feature Effect on Log-odds Output for a Single Observation

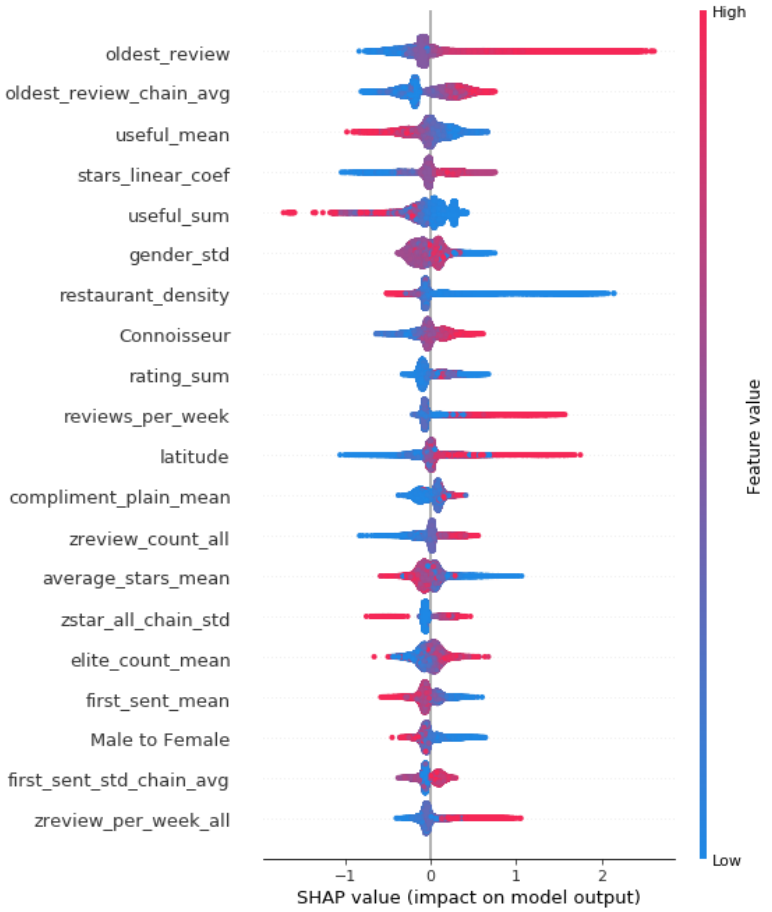


The plots above show an example of the predicted outcomes for two restaurants' observations. The final output is the probability of restaurant closure (0) and the probability of restaurant success (1). As can be seen from the plots, a multitude of features lead to the final predicted outcome. The top observation is predicted to remain open and the bottom observation is predicted to close within the next two years with a decision threshold (rule) of 50%.

Figure 3: Feature Effect on Log-odds Output for a Subsample

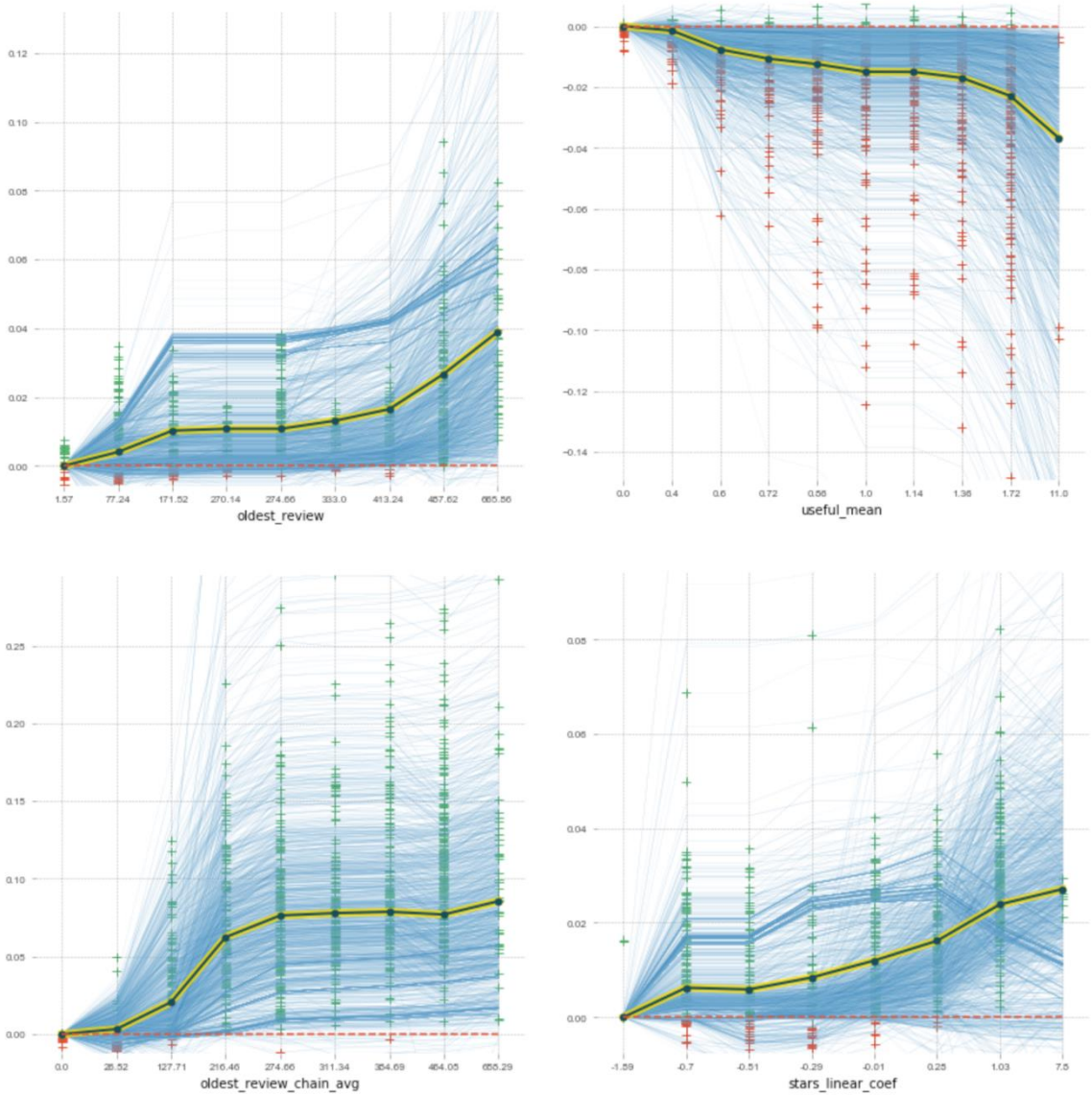
This plot replicates *Figure 2* but for a subsample of 100 firms that are vertically plotted. The samples are sorted by similarity and stacked next to each other. In this plot the predicted outputs are where the blue and red boundaries meet. The variable effects are stacked vertically. The predicted outcome is where the blue and red areas meet. This plot is a close analogue to a DNA strand, the intensity and direction of each of the 430 characteristics determines the final outcome.

Figure 4: Distribution of Individual Feature Effects on Output



This chart reports all the predicted outcomes for each individual variable. It takes on the form of multiple horizontal violin plots, in that it not only reports the effect on outcome but also reports the feature size based on the continuous colour legend and the distribution of the outcomes.

Figure 5: Individual Conditional Expectation (**Depth One**)



These plots point out the non-linear nature the top 4 features. These figures report the marginal relationship of the feature with the predicted outcome. The green marks are open restaurants, the red marks are closed restaurants. The black line (yellow outline), presents the features' marginal effect on the predicted outcome for all observations around the central points on the x-axis. The blue lines are an indication of how all other features further effect the observations to produce the final outcome. (1) Top left is the oldest review of the restaurant in number of days; the older the less likely the restaurant is to close (2) Top right is the number of useful (critical) reviews; the most critical the more likely the restaurant is to close (3) Bottom left is the average age of the oldest review across the chain in number of days; the older the average first review across the chain the less likely the individual restaurant is to close (4) And bottom right measures the slope of historic ratings as measure by stars out of five; the larger the slope the less likely the restaurant is to close.

Appendix

Limitlessness

Many things in machine learning is limitless in the sense that you can tweak it endlessly to achieve some converging performance ceiling. This include the countless ways to do cross validation, to select hyperparameters, to do up-and down-sampling, to perform outlier removal strategies, missing data replacement strategies and many more. Further, features can for example be transformed in an infinite number of ways; the dimensions of features can also be reduced and inflated in innumerable ways; features can be generated through countless unsupervised methods; features can simply be combined, added, or deducted. It is truly limitless.

The question is how do we know if any of these adjustments would lead to a better model? Most of the time we use proxies for potential performance (ppp) like the Akaike information criterion (AIC), or feature-target correlation. These approaches get us halfway towards a good outcome. Another way to go about it is to retest your algorithm once you introduce a new adjustment. The tests can remain in-sample, but for each new test, the cross-validating sections should change or some other form of randomness have to be introduced to ensure that these adjustments do not overfit the validation set. You could also develop two algorithms with different fundamental construction side by side and do constant experimentation; e.g. to know if adjustments should be added to your primary algorithm (XGBoost) test the improvements in your secondary algorithm (CNN). In addition, make sure to perform the correct pre-processing transformations for each algorithm. A cleaner method is to use new data each time you introduce a new adjustment, however, the luxury of a lot of data is hard to come by, even in finance.

Depending on whether you are working with regression or classification problems, you should look at the mean increase in for example accuracy and an increase in the ROC (AUC) score. If the model improves, accept the adjustment. If it is expensive to retrain a full model, create smaller models to do these tests on. If it is still a problem, then resort to the ppp metrics. Do note that the more tests you perform, the more you are abusing the data and increasing the likelihood of overfitting, leading to poor out of sample performance. Finally, one of the best way to automate this experiment testing process is with Bayesian optimisation.

Supervised and Reinforcement Learning

The general pipeline for supervised machine learning trading involves the acquisition of data, the processing of data, the machine learning prediction task, trading policy development, backtesting, parameter optimisation, live paper simulation and finally trading. When there is a shock and regime change, it is better to discard much of the previous parameter weights and focus on new observations (instances, datapoints). Supervised machine learning models, especially neural networks, can keep up with changing market regimes as long as it is able to do online training[1]. Markets appear to be entities that adapt and learns so it is important that the machine learning models also keep on learning to discourage the predictions from becoming stale.

The basic supervised learning task involves price prediction at a point in time. This includes regressors that predict the price level and classifiers that predicts the price direction and magnitude in predefined classifications. From my experience, classifiers generally outperform regressors. What is great about classifiers is that they give you a simple probabilistic interpretation that allows you to easily form deciles and set thresholds for trading strategies. These thresholds are arbitrarily set. The belief is that you would set the thresholds so that you maximise your risk-adjusted profit metric. It is up to the

individual investor to set the rules pertaining to their preferred risk measure (Sharpe Ratio, Max Drawdown, VaR) and other policy considerations.

The benefit of reinforcement learning algorithms is that the final objective function is the preferred risk adjusted score keeping measure. The reinforcement learning process draws on a more elaborate process of decision automation. The reason supervised processes tend to fail is because the itterative process from ML prediction through to policy development, backtesting and parameter optimisation is too slow and fraught with overfitting potholes. This process is slow because the simulation only turns up late in the process after much hard work has already been done. Further, the policy does not develop 'intelligently' with the machine learning model. Let us have a look at how reinforcement learning can help.

Reinforcement learning only has four or so steps as opposed to the seven to eight of supervised learning. RL allows for end-to-end optimisation on what maximises rewards. The RL algorithm directly learns a policy. It is trained on a simulated environment. Like other strategy development methodologies, you still have to separate the backtesting and parameter optimisation step. Remember that the focus should remain on out-of-sample performance at the end of the day, so be wary of over-optimisation and deflate your performance metrics appropriately to control for multiple-testing. Briefly, RL involves data analysis, agent training in a simulated environment, paper trading, and then finally live trading. In each of the last steps, the agent is exposed to an environment.

Unlike supervised models, reinforcement models specify an action as opposed to a prediction, however the decision masks an underlying prediction. The simplest RL-finance approach is a discretion action space with three actions, buy, hold, and sell. There are several possible reward functions, such as realised/unrealised profit and loss. At this point, we would have to ask, 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 less steps, it takes longer to train; they are very computationally intensive to algorithms. RL needs a lot of data, even more so than supervised ML. It can also be expensive to test if you cannot reconstruct a good simulated environment. In such a case, you might have to revert to the real environment when the simulated environment will not cut it; this can be very expensive. Lastly, the bigger the action space the harder it is to optimise an RL agent's performance, not to mention the ridiculously slow optimisation concerning a continuous as opposed to discrete action space.

It is likely that supervised learning would rule finance in the near future. Supervised learning is itself quite flexible and we should expect to see many 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 have for a long time 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 (meta-labelling), bringing the trading policy and trading rules closer to the ML model and to a data-driven level of intelligence. This is done by simply training models in parallel, with one ingesting the results of the other before predicting the policy in the next time step.

In saying that, I simultaneously expect to see a lot of improvement on the RL front so that RL adopts the pros 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 focused on the optimisation of actions, which is and has always been the primary goal. SL and RL algorithms indirectly pick up on well known trading strategies. The gradient step might make the agent buy more of what did the best yesterday, indirectly creating a momentum investing strategy. With both SL and RL you can build different costs into the predictions, should a true positive be as rewarding as a true negative, should a false positive be highly penalised.

Moving on to model technicalities, GBMs can be used in supervised or reinforcement learning algorithms. However, they are definitely not well suited for RL algorithms. For one, you cannot do

partial fitting with GBMs; you have to construct the whole memory and completely retrain the regressor/classifier at each step. This makes it practically infeasible for reinforcement learning. This is unfortunate because GBMs do perform really well n structured (tabled) data and they do not need as much data (at least for now) as is required for neural networks. Over the years, my go-to machine-learning algorithm went from elastic nets, to gradient boosting models to neural networks enabled by TensorFlow and Keras. I have come to believe that with enough attention, the gradient boosting models do not outperform neural networks, even on tabular data. For the general user it would, because with gradient boosting models a lot of the parameterisation weighting happens internally. It is therefore a good off the shelf method. A further point to consider is that GBMs cannot extrapolate beyond training observation extremes. Neural networks can extrapolate more effectively as they can approximate any continuous function within a compact set[1]. As a result, one should be cautious of adding time variables in GBM models, because they cannot extrapolate beyond them. However, because these models struggle to notice seasonality and trends. One can detrend the data, and then add date features like month values, quarter values and day levels for different granularities of seasonality.

When working with reinforcement learning algorithms one comes to understand that convergence to an optimal value is not always 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 number of sample needed to obtain a level of approximate optimality increases with the state and action space. Further, without any assumptions there is no better way than exploring this space randomly, so progress is small and slow. In addition, a last criticism just for good measure, continuous states and actions are a real problem for RL. Therefore, the RL issues boil down to the following: how are we supposed to visit an infinite number of states, an infinite number of times for an infinite number of continuous values with pseudo-random small and slow time steps. Well this can only be done by the generalised nature of supervised learning. Generalisation can be adopted in RL using function approximation as opposed 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 the model off with no data, and as you collect data the action value changes and the ground truth labels don't remain fixed; a point previously labelled as good, might look bad in the longer run. However, notice how great it is that you are never given any samples from the 'true' target function yet you are able to learn by optimising on a goal, that is why RL is intellectually so appealing. To get closer to the true function, the agent has to keep exploring. The exploration in uncertain dynamics means that RL is much more sensitive to hyper-parameters and random seeds than SL as it does not train on a fixed data set and is dependent on network output, an exploration mechanism, and environmental randomness.

Alternative Data

About 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 was introduced to make statistical arbitrage much easier and more accurate, lately we have seen the rise of alternative data strategies, posed to be as important as the previous three 'revolutions'. Although the commercial potential of AI and alternative data is enormous, trading results will side with groups with bigger machines, more innovative data scientists and larger and better datasets.

As much as I hate the recent developments in the area, depending on the size of your organisation, some of the data might be of benefit, not because they produce independent alpha, but because they

provide for important interaction effects with current data. After leaving my morality at the door, I would give special attention to scraped data (see the restaurant example in Quantamental strategies, or follow this data), as well as credit card, email transaction and web traffic data; most other data are overblown; the idea is to get as close to the management accounts as possible, the above are good proxies. I have also heard of a few people that are successful earnings call transcripts.

Recursive Learning:

It is important to be intelligent all the way down. Although one can create reinforcement-learning agents that are 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 upwards. Lower level agents can look at pricing, fundamental and limited capital market data while a meta-agent can select or combine strategies based on potential regime shifts that happen at the economic level. A meta-learner effectively chooses 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 an overall portfolio. The core function is to carry our portfolio level statistical arbitrage to the extreme using all the tools available financial or otherwise.

It is important to understand this concept, all the data do not have to be ingested at the lower levels. Eventually all informative data should be incorporated somewhere in the hierarchical structure. And, to know where becomes an optimisation problem in itself. I will introduce another concept that is controversial but makes economic sense. Once your model is fully developed, introduce a little random gaussian noise at as many levels as possible. When large systematic funds collectively dive, these funds it is because they are too focused on next period performance as opposed to long-term generalisability. Discretionary funds meanwhile do fine because their investment decisions do not follow a particular rationale. Systematic funds should regularise their trading strategies. This is going to become more important as new entrants crowd out the incumbents and the incumbents crowd out themselves.

You need not use an additional level of reinforcement learning algorithm; you can also use clustering algorithms. One can discover multiple economic regimes by simply using a KNN clustering technique to select the potential regime of the last 30 days. Then one can perform strategy selection by looking at the historic success across all regime types. While doing this, pay attention to the stability of clusters. The assignments might not persist in time series; when you face these problems always remember that MiniBatch and Ward clustering algorithms tend to be more persistent.

Tricks and Tips:

- Create a lagged variable of the difference between the past predicted and past true value. Naturally set this value to zero at the start of the process.
- To identify information leakage, or whether data 'peeking' has occurred, look at which feature best predicts whether an observation is in the test or training set.
- Difference between a good and great strategy is not that important, position sizing, risk management, resilience, execution, and infrastructure are key – and stay wary of front running brokers.
- Decide on high value vs low value strategies by considering the strength of your buying power.
- Using an unbiased auto-ml model I have investigated the returns that could be earned using machine learning models over time. The profitability of using general ML and publicly available

data is monotonously decreasing. You either have to join the march down to zero marginal profits or think outside of the auto-ml box – note doing this is already quite a high threshold.

- Discretionary funds should think of a post-ml strategy, where you seek to understand how far reaching these models can be and to reaffirm a strategy that invests beyond an ML-assisted purview.
- Understand that finance always goes through its crazes, from chaos theory to GARCH models. Although ML could generally be revolutionary, I suspect it will mostly be the result of minimising overheard costs as opposed to improving alpha.
- Change nodes in recurrent neural networks to position different levels of importance, not just in terms of time but also by weighting other potential criteria such as volume, whilst still preserving the causal order.
- Never forget outlier detection and Winzorisation of data.
- If you are working with long term data, adjust the values with inflation, otherwise it might lead to some low-level time leakage (yes these models are that smart).
- When PCAs are non-performant, try ICAs, they are better at distinguishing non-Gaussian signals, even better, entropy component analysis.
- Always execute early stopping where possible on a validation set.
- Always know the power of your statistical tests; you might be throwing away many potential true positives.
- Be cautious of trading strategies that produce stable daily profits, there is a good chance that you are engaged in negative skew trading.
- Do not be afraid to pool multiple instruments together, it is an excellent way of getting more data history.
- Do not be afraid to focus on the Sharpe or other risk adjusted performance measures and resorting to leverage to buy returns.
- Bootstrap your optimisation where possible, with that I mean repeat the optimisation many times and average out the resulting weights. This is already quite common in machine learning.
- Discrete mathematics turn up in many ML problems, often times there is no closed form solutions, and the only hope is to use brute force.
- There is at least twenty different cross-validation techniques that I have encountered or created to suit a specific task. One can be very creative with cross-validation techniques, hold two rules in mind when working with values in time series, do not leak too much information and give added importance to more recent results; these two would always be in conflict.
- The next tip might sound juvenile, but do not forget to retrain your model on the entire train set with the newly discovered optimised hyperparameters.
- Unlike some researchers, I believe in the power of backtests to improve models to the point they are used to adjust hyperparameters (some call this model development). However, once you look at the performance on the test data (out of sample back-test), you should not change the model anymore, you should either keep the model or discard the model. If you keep the model, you can retrain and re-optimize your parameters with both the test and train data.
- The less you overfit the less you have to worry about leakage.
- Make sure that your data is not misaligned, pay special attention to fundamental data, the data that should be recorded is the data that was available to the public at that point - also make sure not to include the adjusted amounts, as the adjustments only occurred later.
- PCAs are sensitive to outliers, so Winzorise the data at the 2.5% and 97.5% percentiles. It is also good practice to normalise the data if you have variables with different units of measurement before performing PCA.

Gradient Boosting Model

Supervised learning refers to the mathematical structure describing how to make a prediction $\boldsymbol{y_i}$ given $\boldsymbol{x_i}$. The value we predict, $\boldsymbol{y_i}$, has different interpretations depending on whether we are implementing a classification or a regression task. In classification task prediction, $\boldsymbol{y_i}$, is the probability of an earnings surprise event of some specified threshold. For the regression task, the $\boldsymbol{y_i}$ is the actual EPS value for the quarter. The inputs, $\boldsymbol{x_i}$, have been selected based on selection procedures applied once for each of the models. Apart from the different prediction types, in the classification task, the model gets logistic transformed to obtain a vector of probabilities for each observation and associated categories. In supervised learning, parameters play an important role. The parameters are the undetermined part that we are required to learn using the training data. For example, in a linear univariate regression, $\hat{y}_i = \sum_j \theta_j \boldsymbol{x_{ij}}$, the coefficient θ is the parameter.

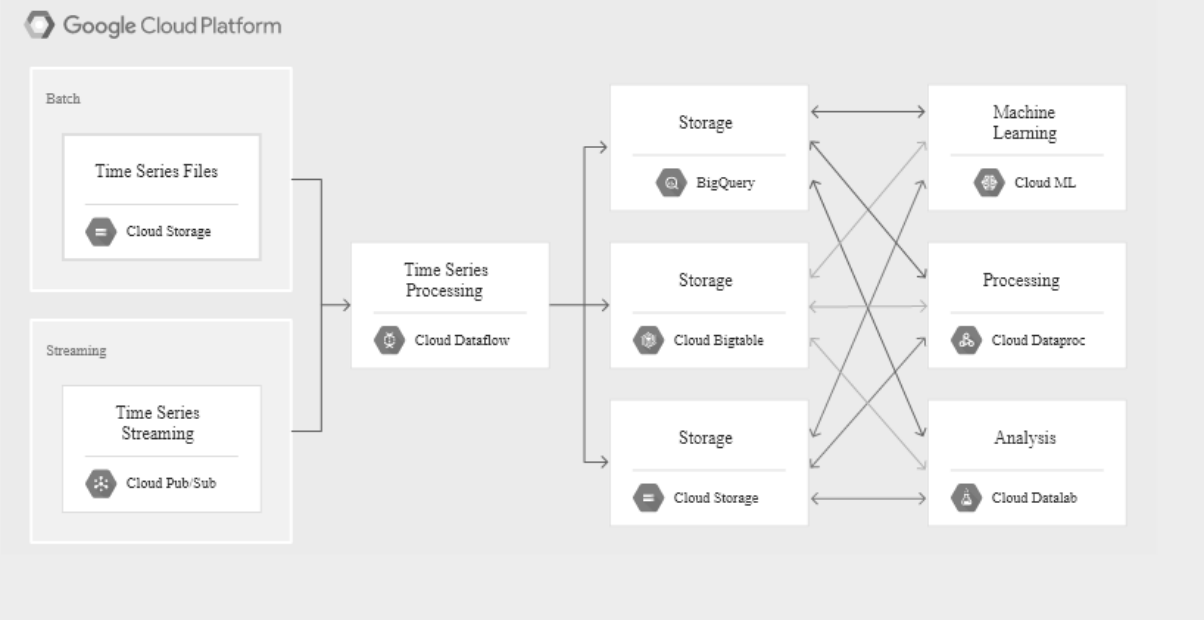
The task is ultimately to find the best parameters and to choose a computationally efficient way of doing so. This choice is largely driven by whether or not we are working with a regression or classification problem. To measure a model's performance, given some parameter selections, we are required to define an objective function. The following is a compressed form of the objective function, $Obj(\Theta) = L(\theta) + \Omega(\Theta)$. In this equation, L is the training loss function; the regularisation term is Ω. The training loss function tests the predictive ability of the model using training data. A commonly used method to calculate the training loss is the mean squared error, $L(\theta) = \sum_i (\boldsymbol{y}_i - \hat{y}_i)^2$. Thus, the parameters get passed into a model that calculates, $\boldsymbol{\hat{y}_i}$, a series of predictions, that gets compared against the actual values in a mean squared error function to calculate the loss. The regularisation term controls the complexity of the model, which helps to avoid overfitting. The Extreme, X, of the XGBoost model, relates to an extreme form of regularisation that controls for over-fitting, leading to improved performance over other models. There are countless ways to regularise models, in essence, we constrain a model by giving it fewer degrees of freedom; for example, to regularise a polynomial model, we can transform the model to reduce the number of polynomial degrees. The tree ensemble can either be a set of classification or a set of regression trees. It is usually the case that one tree is not sufficiently predictive, hence the use of a tree ensemble model that sums the predictions of many trees together. Mathematically, the model can be written in the following form $\hat{y}_i = \sum_{k=1}^{K} f_k(\boldsymbol{x_i})$, $f_k \in F$. Here, *K* is the number of trees, and *f* represents one possible function from the entire functional space *F*. *F* is a set of all possible classification and regression trees (CARTs). This expression then simply adds multiple models together that lives within the allowable CART function space. Therefore, combining the model, the training loss and regularisation function, we can gain our objective function and seek to optimise it, the function can be written as follows, $Obj(\theta) = \sum_i^n l\left(\boldsymbol{y_i}, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{K} \Omega(f_i)$. Thus far, the model is similar to that of a random forest, the difference being in how the models are trained.

For the next part, we have to let the trees learn, so for each tree, we have to describe and optimise an objective function, we can start off by assuming the following function, $Obj = \sum_i^n l(y_i,) + \sum_{i=1}^{t} \Omega(f_i)$. By looking at the function it is important that we identify the parameters of the trees. We want to learn the functions, f_i, each which contains a tree structure and associated leaf scores. This is more complex than traditional methods where you can simply take the gradient and optimise for it. Instead, Gradient Boosting uses an additive strategy, whereby we learn to adjust and add an extra tree after each iteration. We write our prediction value at step t as $\hat{y}_i^{(t)}$, so that we have $\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(\boldsymbol{x_i}) = \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x_i})$. Then we simply choose the tree that optimises our objective, $Obj^{(t)} =$

$\sum_i^n l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^t \Omega(f_i) = \sum_{i=1}^n l\left(\boldsymbol{y_i}, \hat{y}_i^{(t-1)}\right) + f_t(\boldsymbol{x_i}) + \Omega(f_t) + constant$. By using MSE as the loss function, it becomes $Obj^{(t)} = \sum_{i=1}^n \left[2\left(\hat{y}_i^{(t-1)} - \boldsymbol{y_i}\right) f_t(\boldsymbol{x_i}) + f_t(\boldsymbol{x_i})^2\right] + \Omega(f_t) + constant$. The form of MSE is easy to deal with. The Taylor expansion can simply be taken to the second order. $Obj^{(t)} = \sum_{i=1}^n \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(\boldsymbol{x_i}) + \frac{1}{2} h_j {f_t}^2(\boldsymbol{x_i})\right] + \Omega(f_t) + constant$, where g_i and h_i is defined as, $g_i = \partial_{\hat{y}_i^{(t-1)}} l(\boldsymbol{y_i}, \hat{y}_i^{(t-1)})$, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(\boldsymbol{y_i}, \hat{y}_i^{(t-1)})$. After all the constants are removed, then the objective at *t* get transformed to, $\sum_{i=1}^n \left[g_i f_t(\boldsymbol{x_i}) + \frac{1}{2} h_j {f_t}^2(\boldsymbol{x_i})\right] + \Omega(f_t)$. This then becomes an adjusted optimization function for the new tree. Although we have looked at the training step, we have not looked at regularisation yet. The next step is to specify how complex the tree should be, $\Omega(f_t)$. To do this we can improve the tree definition to *F(x)*, $f_t(\boldsymbol{x}) = w_{q(\boldsymbol{x})}, w \in \mathbb{R}^T, q: \mathbb{R}^m \to \{1, 2, \ldots, T\}$. Here *w* represents the scores of the leaves presented in vector form and *q* represents a function that assigns each point to the appropriate leaf, lastly *T* denotes how many leafs there are. The complexity can be defined as $a\ \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2$; there are more ways to formulate and define how complex a model is or should be in practice, but this one is quite practical and easy to conceptualise. Once the tree model is described, the objective value w.r.t. the *t-th* tree can be written as follows: $Obj^{(t)} = \sum_{i=1}^n \left[g_i f_t(\boldsymbol{x_i}) + \frac{1}{2} h_j {f_t}^2(\boldsymbol{x_i})\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2 = \sum_{J=1}^T \left[\left(\sum_{i \in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i \in I_j} g_i + \lambda\right) w_j^2\right] + \gamma T$, where $\mathrm{I}_j = \{\mathrm{i} | \mathrm{q}(\mathrm{x}_j) = \mathrm{j}\}$ represents a full set of all the data points. as have been assigned to the *j-th* leaf. The equation can then further be compressed by describing $G_j = \sum_{i \in I_j} g_i$ and $H_J = \sum_{i \in I_j} h_i$, then $Obj^{(t)} = \sum_{J=1}^T \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2\right] + \gamma T$. In the preceding equation the weights, w_j are independent w.r.t each other, the form $G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2$ is quadratic, and the best weight for a structure *q(x)* is given by the following expression. $w_j^* = -\frac{G_j}{H_j + \lambda}$, $obj^* = -\frac{1}{2}\sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$. This equation measures how good a tree structure $q(x)$ is. A lower score is better for the ultimate structure of a tre. Now that we know how to measure the fittingness of a tree is, we can identify all the trees and select the best one. It is, however, not possible to approach it this way and instead has to be done for one depth level of a tree at a time. This can be approached by splitting a leaf into two sections and then recording its gain. The following equation represents this process, $Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_l + G_l)^2}{H_L + H_R + \lambda}\right] - \gamma$. If the gain obtained is equal to or smaller than γ, then it would be better if we do not add the branch to the tree, this is often referred to as the pruning technique. We basically search for the ultimate split, if all instances are sorted in order, we simply scan left to right to sufficiently calculate the structure scores of all possible solutions and then identify the most efficient split.

Deployment

Once you have developed your model you have to bring it into production. The best way to do it is to tune into an API or develop an API service yourself. You can learn to do this online. Here is an API applied in for a risk management task as an example. There are some great deployment articles out there. I am myself a fan of the Google Cloud Platform. They allow you to set up virtual machines with on demand access to storage and compute and I can easily connect into my current storage environment. The following image will give you a reasonable idea of how you can set up your cloud operations in a finance domain.



Finance and Machine Learning

Each industry will be touched by machine learning. In the future, I would go down this mapping to identify all the financial domains bound to be changed by recent advances in machine learning.

1. Corporate and Retail Banking
2. Investment Banking
3. Insurance Services
4. Payment Services
5. Financial Management
    - Public Finance
    - Financial Economics
    - Management Accounting
6. Advisory Services
    - Personal Finance
    - Consulting
7. **Investment Services**
    - Private Equity and Venture Capital
    - Wealth Management
    - Asset Management
    - **Broker Dealer**
        - **Liquidity**
            - Market Making
            - **Predicting Customer Needs**
                - ***Recommender AI System***
            - Shocks and volatiliy
        - Strat:
            - Extreme Risk
            - Simulation
        - Capital Management

Other SSRN Papers

- Financial Machine Learning Regulation
- Predicting Restaurant Facility Closures
- Predicting Corporate Bankruptcies
- Predicting Earnings Surprises