Ensemble learning applied to quant equity: Using gradient boosting in a multi-factor framework

Tony Guida & Guillaume Coqueret

Abstract. In this chapter, we apply a popular and Kaggle winner Machine Learning approach (extreme gradient boosted trees) to build enhanced diversified equity portfolios. A simple naïve equally-weighted portfolio of US stocks based on a boosted tree-based signal generates on average an excess return of 3.1% per annum, compared to a simple multifactor portfolio. We demonstrate that using boosted trees on a large number of features give an average error rate of 20% for predicting the 12-month sector neutral outperformance of a stock. In addition, enhancing a simple multi-factor signal with an ML-boosted signal proves to add value on a risk return basis without altering the factor exposure of the traditional multi-factor portfolio.

1. Introduction

It is a both intuitive and well-documented fact that firm's performance on the stock market is driven by some of their core characteristic. In their seminal article, Fama and French (1992) show that firms with higher book-to-market ratios significantly outperform those with low book-to-market ratios. They also report that small firms also tend to yield returns that are higher than those of large firms. [1] Later, Jegadeesh and Titman (1993, 2001) constructed abnormaly profitable (momentum) portfolios buy buying outperforming stocks and shorting underperforming ones.

Findings such as these ones have led to the construction of so-called factor indices in which the investor buy the above-average performing stocks and sells the below-average ones. The literature on these *anomalies* is incredibly vast and has its own meta-studies (see e.g., Subrahmanyam (2010), Green et al. (2013) and Harvey et al. (2016)). [2]

It can be debated whether these discrepancies in performance originate from truly pervasive (and priced) factors that structure the cross-section of stock returns (a stream of literature that was launched by Fama and French (1993)), or from the firms' characteristics directly, as put forward by Daniel and Titman (1997).

In any case, there seems to be a large consensus that investors should be able to benefit from the introduction of firms' characteristics in their asset allocation process. This seemingly obvious advice is all the more pertinent since smart-beta

[1] This is usually referred to as the size premium. This stream of literature was initiated by Banz (1981) and is reviewed in Van Dijk (2011).
[2] In addition, McLean and Pontiff (2016) shed some light on this topic through the lens of predictability.

indices are reshaping the asset management industry (Kahn and Lemmon (2016)). Beyond simple portfolio construction processes[3], more sophisticated methods have emerged, for instance in Brandt et al. (2009) and Ammann et al. (2016).

The rise of Artificial Intelligence (AI) and more specifically to Machine Learning (ML) in unrelated fields (computer vision, translation, etc.) has had an impact on how quantitative managers can process all of the data they have at hand. Recent contributions encompass techniques such as Bayesian inference (Bodnar et al. (2017)), flag pattern recognition (Arévalo et al. (2017)), clustering (Nair et al. (2017)), random forests, boosted trees and neural networks (Ballings et al. (2015), Patel et al. (2015) and Krauss et al. (2017)) or even recurrent neural networks (Fisher and Krauss (2018)).

The limitation of most articles above is that the predictors are usually limited to price data or possibly technical data. This is suboptimal because as the asset pricing literature has demonstrated, there are many other candidates for explanatory variables. In this chapter we propose to benefit from the advantages of machine learning in general and boosted trees in particular, e.g. non-linearity, regularization and good generalization results, scaling up well with lots of data. The present contribution is closest in spirit to the work of Ballings et al. (2015). The main difference between the two lies in the sophistication of the labelling process: Ballings et al. (2015) simply look at price direction, while we take a more structured approach.

This chapter is organized as follows. In Section 2, we give a mildly technical introduction to boosted trees. Section 3 is dedicated to data and protocol and will introduce the construction of the dataset with the feature and labels engineering, the protocol that we will use in the subsequent section and the calibration of the ML applying rigorous protocol established by the computer science community.

2. A Primer on boosted trees

This section is dedicated to a self-contained and reasonably technical introduction on decision tress and boosted trees. For more details, we refer to Chapters 9 and 10 of Friedman et al. (2009).

We consider a database that is split in two: the explanatory variables, gathered in the matrix $\boldsymbol{x}$ and the variable we aim to forecast, which, for simplicity here, we assume to be a vector, y. Let T be the number of occurrences in the data and K be the number of explanatory variables: the matrix $\boldsymbol{x} = x_{t,k}$ has dimensions $(T \times K)$. Henceforth, we note $\boldsymbol{x_t}$ for the K-valued vector containing all fields of occurrence t.

The purpose of the tree is to partition the data (i.e., the collection of $(\boldsymbol{x},\boldsymbol{y})$) in clusters in which the elements y_t are as similar as possible. If $\boldsymbol{y}$ is a numerical

[3] For a more detailed view of the intertwining between factor investing and asset management, we refer to the monographs of Ilmanen (2011) and Ang (2014).

variable, this means reducing the variance inside the cluster and if it is a categorical variable, then it amounts to reduce the "impurity" of the cluster (we seek a strongly dominant class).

To ease the presentation, we deal with regression trees first. At the root of the tree, the optimal split *s* for variable *j* is such that the two clusters formed according to this variable have the smallest total variance in $\boldsymbol{y}$:

$$V_j^s = \sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}>s\}}(y_t - \mu_j^+)^2 + \sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}\leq s\}}(y_t - \mu_j^-)^2,$$

where μ_j^+ and μ_j^- are the intra-cluster averages:

$$\mu_j^+ = \frac{\sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}>s\}} y_t}{\sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}>s\}}}, \quad \mu_j^- = \frac{\sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}\leq s\}} y_t}{\sum_{t=1}^{T} \mathbf{1}_{\{x_{t,k}\leq s\}}}.$$

The notation $\mathbf{1}_{\{.\}}$ denotes the indicator operator: $\mathbf{1}_{\{x\}}$ is equal to one if *x* is true and to zero if not. For all explanatory variables *j*, the algorithm minimizes V_j^s across all plausible values *s* and retains the one for which the total variance is the smallest. The first split is then performed and the procedure is repeated on the two resulting clusters.

Note that in the definition of V_j^s, the terms $(y_t - \mu_j^{\pm})^2$ are simply scaled variances because we build a regression trees. The analogy with linear regression is straightforward: the classical OLS estimator also seeks to minimize the variance between the actual data and the predicted values. In the case of classification trees, the computation of the variance is replaced by a metric that captures the impurity of the cluster. One popular of such measures is the cross-entropy. If $\pi_k^{s\pm}$ are the $K^{\pm}$ proportions of the classes of $\boldsymbol{y}$ in the two clusters resulting from the sort *s*, cross-entropy is a common measure of impurity: $-\sum_{k=1}^{K^{\pm}} \pi_k^{s\pm} \log(\pi_k^{s\pm})$. Minimizing the cross-entropy usually leads to the emergence of one dominant class (at least, that is its purpose).

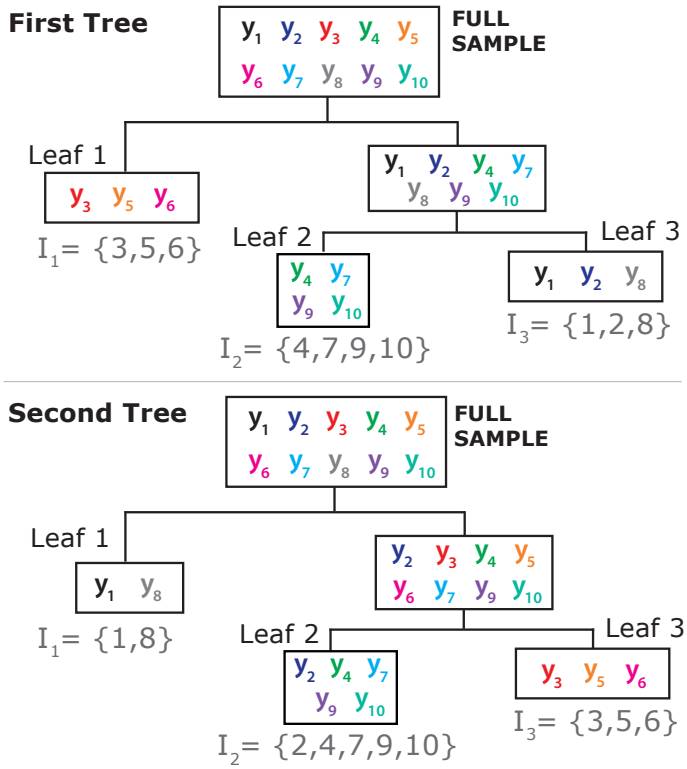
The tree progressively grows when nodes are split in two and the fit naturally increases with the number of leaves. Obviously, a tree with hundreds of leaves is likely to overfit the data. The criterion for fixing the number of nodes is usually a linear combination: goodness of fit minus a penalizing term consisting of a multiple of the number of leaves.

Once one tree is built, the idea behind boosting is to combine it with one or many other trees to increase the goodness-of-fit (this is a particular case of ensemble learning). An intuitive solution is to train several classifiers and to combine their predictions into one output signal. In his seminal contribution, Schapire (1990) proposes to fit three trees and to then use a majority vote for a binary classification. Refinements of this idea led to the development of the family of AdaBoost classifiers (Freund & Schapire (1997)). We refer to Friedman et al.

(2000) for a review on this topic. In the latter papers, the authors show that the Adaboost principle admits a simple additive representation.

To graphically illustrate these ideas, we plot to simple trees in Figure 1. We are only interested in the determinant feature, i.e., $\boldsymbol{y}$. The values of the latter are coded through colors and the purpose of the tree is to build clusters with similar colors. Both trees end up with a 'hot' cluster (red-orange-pink), but they differ on where to locate the second instance y_2 (dark blue). Now, if we were to predict the color of a new occurrence with features like those of y_2, our prediction would mix the outcomes of the two corresponding clusters.

FIGURE 1: **Two symbolic trees**. Variations in the dependent variable ($\boldsymbol{y}$) are represented with colors. The black rectangles and segments show the structure of the tree. The I_j in grey are the instance sets of the leaves.



We now specify the additive method in more depth. Let us start with one fitted tree and let's add another tree "on top of it" that reduces the errors of the first tree (e.g., by fitting the new tree to the residuals). Let us call T_1 the first tree.

The second tree T_2 is built in the following manner: $T_2(\boldsymbol{x}_t) = T_1(\boldsymbol{x}_t) + \gamma_2 f_2(\boldsymbol{x}_t)$ where γ_2 and f_2 are chosen so that T_2 minimizes the loss function (total variance or

weighted sum of cross-entropy for instance). The procedure can of course be iterated any number of times:

$$T_m(\boldsymbol{x}_t) = T_{m-1}(\boldsymbol{x}_t) + \gamma_m f_m(\boldsymbol{x}_t).$$

The true challenge is obviously to find the optimal γ_m and f_m. Recent approaches[4] tackle this problem using gradient-based techniques. Below, we describe the algorithm behind XGBoost (Chen and Guestrin (2016)). For each occurrence, the method boils down to the computation of the weighted sum of prediction stemming from the different trees.

We start with some notation. We write $\hat{y}^m$ the prediction of the m^{th} iteration of the process. L is the loss function: e.g., weighted variance for a regression tree or weighted cross-entropy for multi-class classification. The objective we seek to minimize is the following:

$$\Lambda^m = \sum_{t=1}^{T} L\left(y_t, \hat{y}_t^m + f^m(\boldsymbol{x}_t)\right) + \Omega(f^m),$$

where f^m is the function (here, the tree) we are seeking. $\Omega(f^m)$ is a regularization term that penalizes the complexity of the tree. We abstractly write q for the structure of the f^m (nodes / splits). In addition, we set, without loss of generality, the number of leaves to J and their weights (in the final weighted sum) to w_j. Assuming an L^2 form for $\Omega(f^m)$ and using a second order Taylor expansion of L with respect to $\hat{y}_t^m$, the objective simplifies to the approximate form

$$\tilde{\Lambda}^m = \sum_{t=1}^{T} \left[g_i f^m(\boldsymbol{x}_t) + \frac{1}{2} h_i f^m(\boldsymbol{x}_t)^2 \right] + \frac{\lambda}{2} \sum_{j=1}^{J} w_j^2 \,,$$

where g_i and h_i correspond to the first two derivatives in the Taylor expansion. If we define the instance set of leaf number j: $I_j = \{i | q(\boldsymbol{x_t}) = j\}$, then

$$\tilde{\Lambda}^m = \sum_{j=1}^{J} \left[w_j \sum\nolimits_{k \in I_j} g_k + \frac{w_j^2}{2} \left(\sum\nolimits_{k \in I_j} h_k + \lambda \right) \right],$$

and the minimizing weights are, for each given leaf:

$$w_j^* = -\frac{\sum_{k \in I_j} g_k}{\sum_{k \in I_j} h_k + \lambda}.$$

The question is then to find a proper tree structure and this is usually performed via some greedy algorithm. Note that in the weights above, the gradient sits at the

[4] For instance, XGBoost and LightGBM; and both are based on the seminal idea from Friedman (2001).

numerator, which seems intuitive given the negative sign: as is customary, the algorithm goes in the opposite direction. Finally, refinements can be incorporated to further enhance the algorithm. One such possibility is shrinkage. The idea behind it is that full-scale learning can lead the optimization in the good direction, but *too far*.[5] Hence, newly added trees can be slightly diluted by a factor η, which leaves more room for future trees:

$$T_m(\boldsymbol{x}_t) = T_{m-1}(\boldsymbol{x}_t) + \eta\, \gamma_m f_m(\boldsymbol{x}_t).$$

Another possibility is subsampling and we refer to the original contributions for more details on this topic.

3. Data and protocol

This section describes the data used and the empirical protocol for our ML model. We focus on US stocks in order to avoid dealing with different currencies and countries as we might find in European or global stocks. We also selected the universe of US stocks for its higher coverage of financial metrics and its relative efficiency.

Henceforth, we use interchangeably the term "feature" or dependent variable to express a stock characteristic. In this section, we will explain the features transformation that has been performed to linearize each characteristic and to express them in the same unit (even if XGboost and tree regression are designed to cope with non-normalized variables).

3.1. Data

We collect monthly returns and monthly stocks' characteristics for the top 3000 US stocks according to their market capitalization, free-float adjusted. The full dataset goes from December 1999 until December 2017. The universe of stocks consists of all common equities using Quandl premium equity packages. The dataset is point in time and therefore do not suffer from survivorship bias. Prices are monthly discrete total return, taking into account stocks splits and dividend adjustments. Prices are expressed in dollars as all the others amounts.

This dataset represents approximately 620,000 instances, an instance consisting of the combination of a stock and a date. The variable y we want to predict is the probability of 1-year forward sector-neutral outperformance. The explanatory variables in our model encompass a large set of 200 features based on traditional, financial, price and volume-based metrics.

---

[5] We invite the interested reader to visit the Kaggle blog for a deep dive into hyper-parameters tuning. http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

In order to avoid a look ahead bias we will use a 24 months rolling windows for training the model. Therefore, prediction will only be possible at t+12 months, and we offset the prediction date by the forward time period used for training. We will repeat the training every month, hence updating every month the probability for each stock to outperform after 12 months. Each rolling analysis period will be split according to 80% training data and 20% of testing data, keeping the testing data in the most recent part of the rolling window to avoid "testing in the past". The testing part is used to adjust the hyper-parameters because it is paramount to avoid overfitting in order to produce outperformance out-of-sample.

3.2. Features and labels engineering

A substantial portion of research in ML-based financial applications fails because of a lack of economic framing and unrealistic or ill-defined goals, such as finding the "best stocks". Instead, our purpose is more reasonable, as we seek to predict extreme behavior and single out the goods stocks from the worst ones within each sector and express this as a probability in order to rank the full cross section of stocks.

We "engineer" both labels (future returns) and features so as to put a structure that will provide the algorithm with a more causal representation of the equity markets. Again, we shift away from the traditional approach that seeks to infer future performance from past prices or short term returns. We set fundamental, risk, volume and momentum based signals as our features. Each feature and label is expressed in z-scores and then translated into percentile to ease the comparison in the results analysis part.
Following the old quant saying "*garbage in, garbage out*" we try as much as possible to impose some structure to features.

In the same fashion, we impose some structure in the labels by sequentially:

1. Resorting to one year (1Y) performance, which is enough for having a certain degree of causality between the nature of the features in the datasets and the tenor of the labels.
2. Normalizing according to the sector of each stock. An alternative would be to use dummy sector variables in the features, but the purpose is much clearer by putting the right structure on the labels.
3. Getting rid of the outliers in the labels: stocks outside the [5^{th}; 95^{th}] percentile of their sector neutral performance are excluded for the training. Our goal here is to imply as much causality for the features with the labels. For instance we are getting rid of stocks that have been acquired in an M&A, or stocks that have been in fraud accounting scandals, because we want the labels to be truly linked to the features.
4. Processing only the remaining top and bottom quintiles of the filtered stocks. We want to approximate a function for the top and bottom parts of the cross section. By doing so, we hope to have a clear hierarchical representation of sector under/out-performing stocks.

We define Y_1^i as the probability of a stock i to outperform its sector S over a one year ahead period. Accordingly, $Y_0^i = 1 - Y_1^i$ will be the probability of a stock i to underperform its sector after one year. Y_1^i serves as the primary input of our classification task. The label we process in the algorithm is the following:

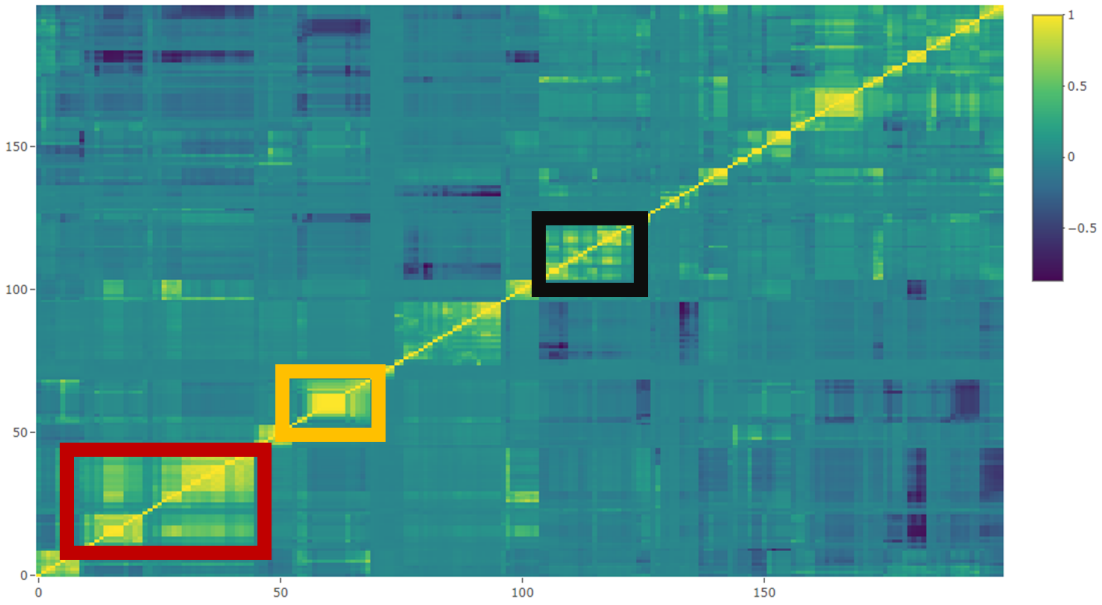
$$y^i = \begin{cases} 1 \; if \; Y_0^i \geq 0.5 \\ 0 \; if \; Y_0^i < 0.5 \end{cases}.$$

Hence, this variable tracks whether or not the corresponding stock is likely to outperform. In the next subsection, we focus on the explanatory variables that we rely on to predict y^i.

3.3. Variables/features used

In our model, we aim to predict, each month, the probability of a stock to outperform its sector using extreme gradient boosted trees. Since we want to sequentially create weak learners (individual trees) and use the residuals (badly classified labels) for the next round, we will use all features in our dataset. In the case of ML prediction using trees, highly correlated variables will not perturb the models. A large number of highly correlated variables will give the algorithm more degrees of freedom to determine the added value of each single variable.

FIGURE 2: **Hierarchical clustering for rank-correlation between variable**. Rank-correlation is computed for the entire dataset and clustered afterwards using Euclidian distance. The hierarchical heatmap graph is color-coded as follows: the more (less) saturated towards yellow (blue), the more (less) correlated.



In order to assess the potential level of correlation between the features datasets we computed a hierarchical clustering for the rank correlation of the features. As depicted in Figure 2, we can identify different groups of metrics that represent

family of signals. For instance the red rectangle in Figure 2 shows the metrics based on valuation ratios, from simple earnings yield and book to price metrics to more rules-based composite metrics, imposing more conditionality depending on the nature of the company.
The yellow rectangle represents the cluster for risk signals based on prices, such as different tenor for price volatilities signals, or correlation acceleration in volatility. In total, the 200 features can be clustered into six families of metrics and we list them in Table 1.

In this chapter, we keep all features in the dataset. Said differently, we do not resort to important feature discovery in the first phase, but rather leave the tree-boosted model to determine which features matter through the regularization parameter in the training part. Moreover, we are using a very short period of time for each step of the training (2 years), hence keeping a high number of characteristics is a good way of having more degrees of freedom when adapting to changing market conditions, e.g., sector and style rotation, risk on-risk off periods, etc.

TABLE 1: **Summary and examples of features per family type**. According to hierarchical cluster six main family of features based on metrics' types. We provide some examples for each family.

Valuation	Prof. / Qual	MoM./ technical	Risk	Estimates	Volume/ liquidity
Earnings Yield	ROE	12-1 monthly returns	5-y bear vol.	EPS revision	Market cap
Book yield	FCF/Assets	6 Months RSI	3-year correlation	EY FY1	Volume
Sales yield	GrossProfit/Capital employed	12-1 M. returns / vol.	Specific risk residual from PCA	EPS growth FY1	Liquidity at Risk

4. Building the model

In the previous section, we presented and explained the objective of the method, the dataset and variables, and how they were structured. We now dig into the details of the general parameters and hyper-parameters used in the xgboost[6] model. In this Section, we introduce the machine learning model, as well as its hyper-parameters that we found of interest using our data. Additionally, we will cover how to tune them in order to give a more practical "how to" to the reader.

Xgboost is an open source model available in different language (C++, R, Python, Julia, Scala) that has been extremely popular in the computer science community thanks to its flexibility in hyper-parameter tuning and to its fast code execution.

[6] Xgboost (eXtreme Gradient Boosting) is an open source package, often referred as the 3rd generation of tree boosting model. The interest reader could find documentation, codes and example on the official website http://xgboost.readthedocs.io/en/latest/

We already covered the mathematical aspects of tree boosting in Section 2, therefore we will restrict the scope of this section in the practical side of things. Our goal in this exercise is to predict the probability of sector neutral outperformance for a stock and we rely on a classification approach (we recall that our labels y^i can only take one or zero as values).

In order to obtain a probability of sector-neutral outperformance, we resort to logistic-based classification: the score of the occurrence will be processed through the sigmoid function[7] which will result in a figure between zero and one.
The objective function will be the usual logistic loss function complemented with a regularization term which we use to control the model complexity. Controlling for model complexity is a first order point for boosted trees as they tend to overfit the data and could exhibit poor generalization behavior out of sample.

4.1. Hyper-parameters

There are many different hyper-parameters in tree boosting, covering them all is outside the scope of the chapter (they often depend on the method of tree-aggregation and on the implementation). We will confine our introduction to the parameters that we have tested or used along this exercise. The list is the following:

- the learning rate, η: it is the step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features and η actually shrinks the feature weights to make the boosting process more conservative.
- the minimum split loss, γ: it is the minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be (trees will be smaller).
- the maximum depth: it is the longest path (in terms of node) from the root to a leaf of the tree. Increasing this value will make the model more complex and more likely to be overfitting.
- the scale of positive weights controls the balance of positive and negative weights: it is useful for unbalanced classes. A typical value to consider: sum(negative cases) / sum(positive cases).
- regression λ: it is the L^2 regularization term on weights (mentioned in the technical section), and increasing this value will make model more conservative.

4.2. Cross-validation

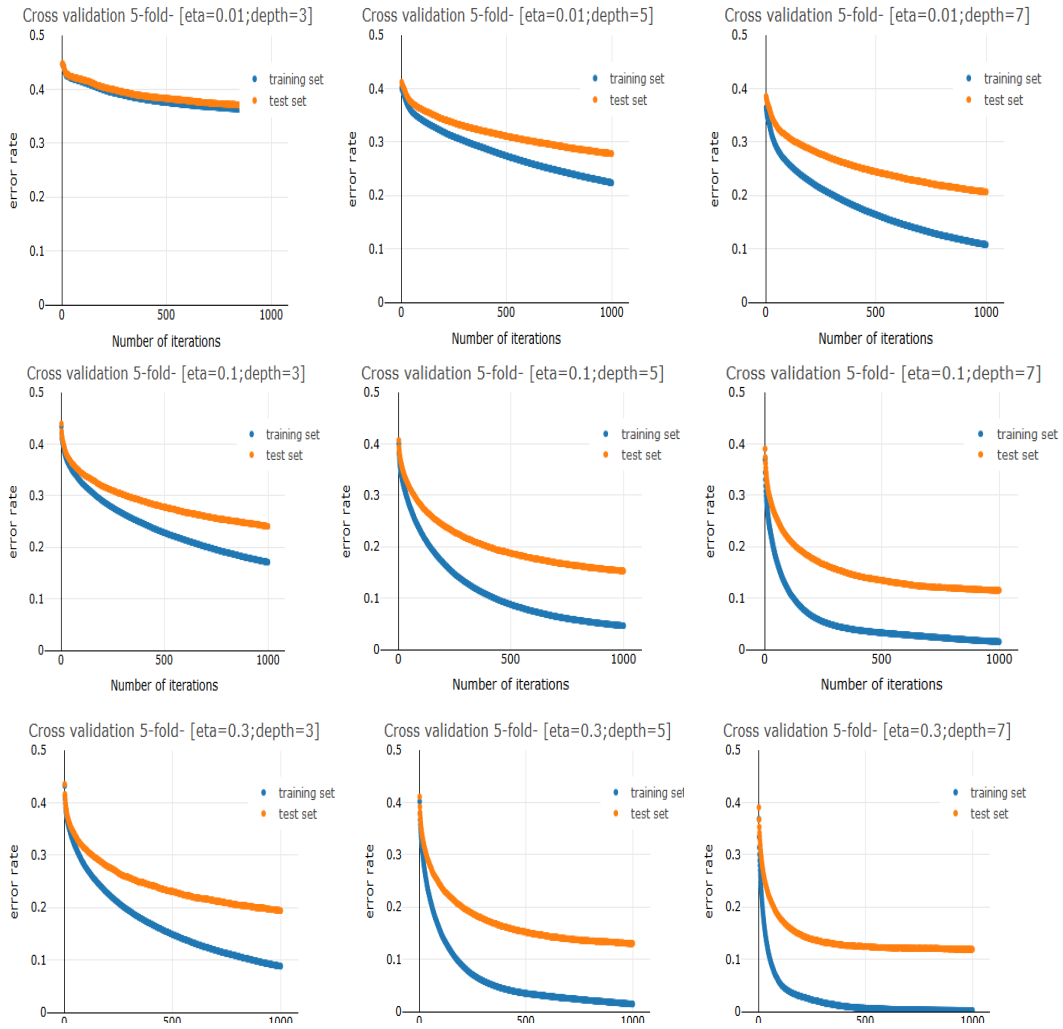
In Figure 3, we perform cross-validation on three different parameters.[8] In order to give to the reader a step by step approach, we computed a chart keeping the training and the test prediction error for each pair of parameters tested on the

[7] The sigmoid function is defined by $S(x) = (1 + e^{-x})^{-1}$.
[8] For more details on cross-validation, we refer to Chapter 7 of Friedman et al. (2009)

aggregation of 1000 trees. The evaluation metric used for this cross validation exercise is the simple mean error, defined by the probability threshold of 0.5, giving the binary classification error rate.
From the left to the right we are increasing the depth of the trees, making them more complex following the sequence of [3;5;7]. From the top to the bottom we are increasing the learning rate from 0.01 to 0.1 to finally 0.3. A higher learning rate means that the model will learn faster and potentially overfit and will not generalize well when predicting on unseen instances.

The bias / variance tradeoff is at the core of the machine learning algorithms and echoes the core principle of the penalized objective function in xgboost: minimizing loss and controlling complexity. A higher error rate associated to a simpler model is more likely to generalize well out of sample. As an example, the model tested with a low shrinkage (eta = 0.01) and very shallow trees (depth = 3) does learn extremely slowly, even after a 1000 iterations. This model, which is the one on the top left part of Figure 3, clearly underfits the data: it is not learning fast enough. On the contrary the model in the bottom right part of Figure 3 (depth = 7; eta =0.3) is learning fast (reaching 20% error rate after 100 rounds in the test set) and is plateauing afterwards. In this example the model is more likely to overfit: this model reaches almost a 99% accuracy in the training set.

FIGURE 3: **5-fold cross-validation for tree boosted models**. We maintain all default parameters except, number of rounds, depth of the trees and learning rate.



Generally speaking, one can see that increasing the depth of the trees helps decreasing the error for a lower level of shrinkage. One can note that for an eta of 0.3 the difference in test error between a depth level of 5 or 7 is very marginal, which suggests some bias in those two models (they managed to reach 99% of accuracy in the training set after a 1000 rounds).

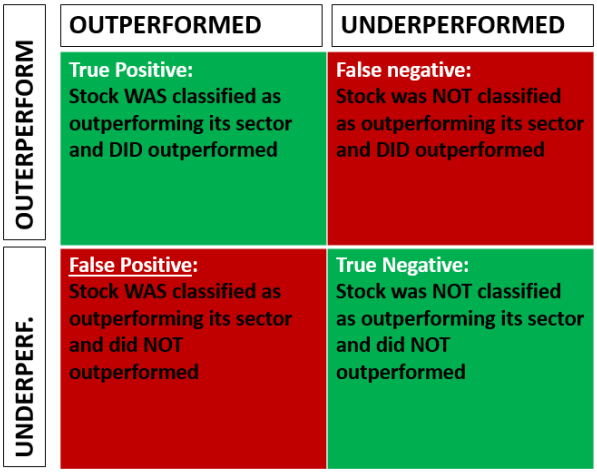
We performed a grid search in order to confirm our conclusions drawn from Figure 3. The parameters selected for our predictive boosted tree model are:

- 1000 rounds with an early stop at 100 in order to prevent overfitting.
- η set at 0.1 to ensure a reasonable learning pace.
- γ set to 0: in our test γ seemed to be of inferior importance, compared to the other parameters.
- Depth of 5: we need to have some (but not too much) complexity to benefit from the full set of 200 features.
- L^2 regularization parameter fixed at 1 which is the default value in the xgboost model.

4.3. Assessing the quality of the model

In the process of assessing the quality of the model, many different evaluation metrics are available. In the cross-validation part, we deliberately only disclosed the mean error for the training and test datasets. In this sub-section, we want to introduce the concept of confusion matrix and all the related metrics in order to precisely assess a machine learning model's quality.

FIGURE 4: **Confusion matrix illustration**. We explain the confusion matrix in our exercise which is a supervised classification model to predict sector neutral outperforming stocks. On the y axis the real labels, on the x axis the predicted labels.



Each part of the Figure 4 can be explicated as:

- **Fp** : false positive. Stock predicted to outperform and that did not outperform out of sample.

- **Fn** : false negative. Stock predicted to underperform that outperform out of sample.
- **Tp**: true positive. Stock predicted to outperform which outperform out of sample.
- **Tn**: true negative. Stock predicted to underperform which underperform out of sample.

From those 4 cases, we can derive several classical metrics that assess the quality of the model.

Precision: **Tp / (Tp + Fp)**
Precision could be defined as a rate of successful prediction for sector neutral outperforming stocks.

Recall: **Tp / (Tp + Fn)**
Recall could be defined as a true rate, since we include the instances that have been wrongly classified in negative.

Accuracy: **(Tp + Tf) / (Tp+Tn+Fp+Fn)**
This is the accuracy level used in the cross validation part.

Those measures can help detecting imbalances in classes, that could lead to a "lazy" classifier problem, where the global accuracy results are good but one class is under represented and showing a lower level of accuracy. In our exercise, we will be less interested in having a great accuracy in finding true negative than true positive.

In our selected model the outcome for the different evaluation metrics are the following

- Accuracy: 0.80
- Precision: 0.797
- Recall: 0.795

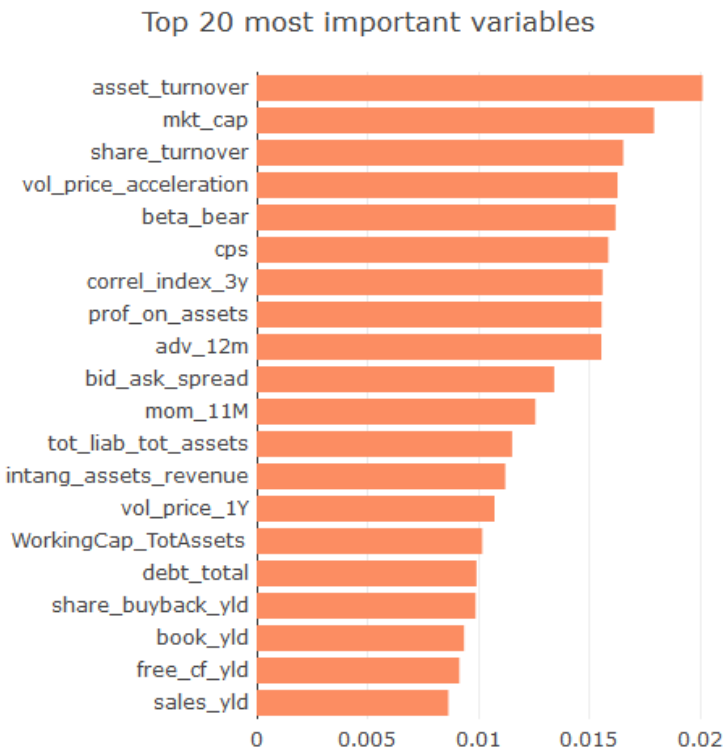
At early stages, we decided to train on the tails of the cross sectional distribution, hence there is very little imbalance in the class: recall, precision and accuracy are therefore very close.

4.4. Variable importance

One common criticism against machine learning is the so-called "black-box"[9] nature of the prediction, as if it was impossible to understand or trace which features or combination of features are responsible for the forecast. Ensemble learning using trees does have a very nice feature that rules out this criticism: variable importance.

[9] Criticism that is not always justified for more complex models, e.g.. Neural nets, that can be "white boxed" with 20 lines of Python.

FIGURE 5: **Top 20 most important variables**. We show the most important variables in our model. We averaged the monthly results for the gain measure after training.



In Figure 5, we display the average variable importance of our model that we trained and used for prediction every months from December 2002 until December 2017.[10] Each month, we keep the variable importance from the trained model. There are a lot a different metrics for variable importance. Popular metric in trees ensemble are the Gini impurity index used for selecting split points.

In our exercise, we use the gain metric which is equal to the relative contribution (in terms of accuracy) to the model from the corresponding features. To compute the gain metric one has to take the contribution for each features for each tree averaged for each month. One can summarize the gain metric as a prediction usefulness indicator. All gain measures across features sum to 1.

First, we can see that on average there is not a high concentration in few features explaining the majority of the prediction's importance. Then, looking at the type of features one can note that:

- We do have features coming from the 6 different metrics families gathered in Table 1.

[10] In order to clarify the protocol: we make predictions every month, e.g. for the last prediction of end of November 2017, we used the features matrix as of Nov 2017 and use the model that has been trained using a 24 month dataset that started in Nov 2014 until Nov 2016.

- Within the top 20 features, price-risk metrics seems to have a better ranking as group than valuation, liquidity metrics etc..
- We do find some very common, well-known and over-researched characteristics mentioned in the asset pricing literature (book-yield for value, market cap for size, profitability on assets for quality, price volatility for low volatility anomaly and 12-1 month momentum).

5. Results and discussion

We now proceed to a use case. Our use case will test our ML-based signals as a base for constructing equally-weighted portfolios. We process our probability of sector outperformance just like any other signal. We normalize it, express it in percentile and assess the performance of monthly rebalanced decile portfolios.[11] As benchmark, we construct 2 signals and follow the same protocol mentioned above. Those two signals are:

1. A simple multi-factor signal blending using commonly accepted composites metrics to reflect the definition of "factor investing"
2. A linear combination of the top 20 metrics, that the most important in our gradient boosting tree model.

In this section, we provide a statistical evaluation of the signal implemented as a naïve strategy.
We will use as benchmarks, an equally-weighted (EW) portfolio made of commonly accepted stock characteristics, which are:

1. **Value**: earnings yield, book yield, EV/EBITDA.
2. **Quality**: return on equity, debt/equity.
3. **Momentum**: 12-1 total return performance.
4. **Low volatility**: 3 years and 1 year price volatility.
5. **Size**: market cap.

The second benchmark will be an equal weight portfolio, using the signal made of a linear combination of the top 20 most important features.

5.1. Time series analysis for equally-weighted decile portfolios

Our purpose in this back-test is to assess the added value of using a machine learning signal in a multi-factor framework compared to existing methodology. In order to compare the different signals we create equal weight decile portfolios according to the ranked z-score of each signal. We then analyze the computed time series of those signals using monthly returns. We finally focus on the top decile (D10, the most tilted) to give more analytical results.

[11] Such portfolio sorting procedures are commonplace since the seminal work of Fama and French (1992).

FIGURE 6: **Wealth curve for decile portfolios based on Multi-Factor signal**.

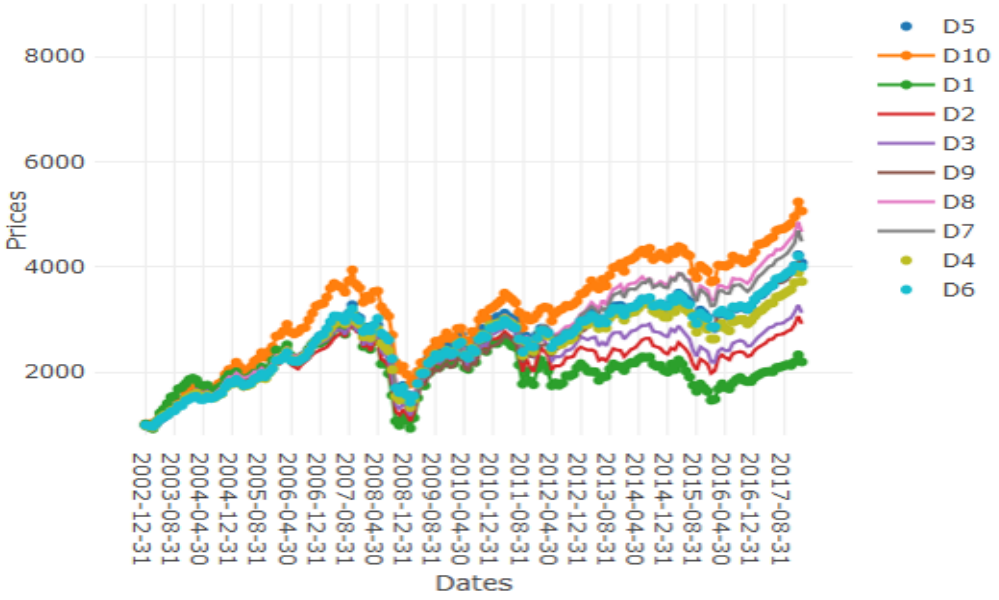


FIGURE 7: **Wealth curve for decile portfolios based on linear combination of the top 20 features from the ML model**.

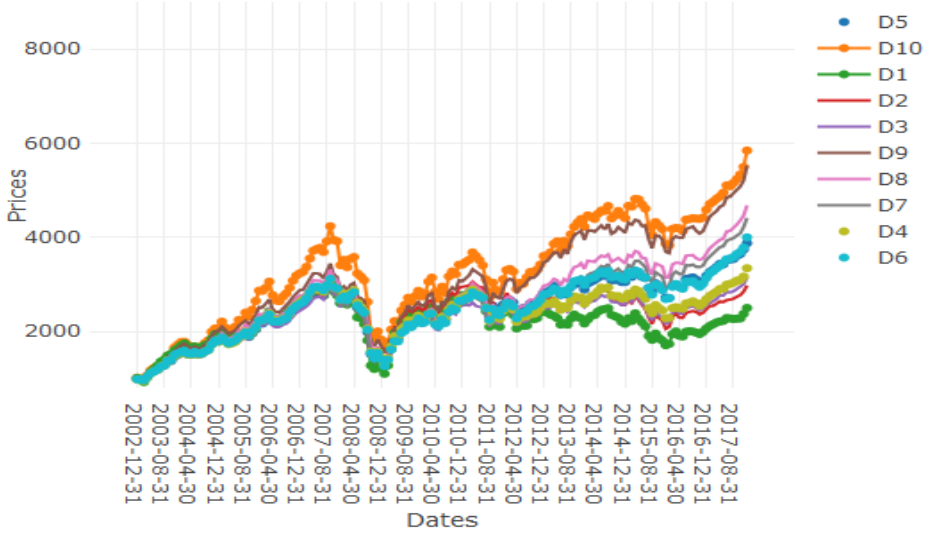
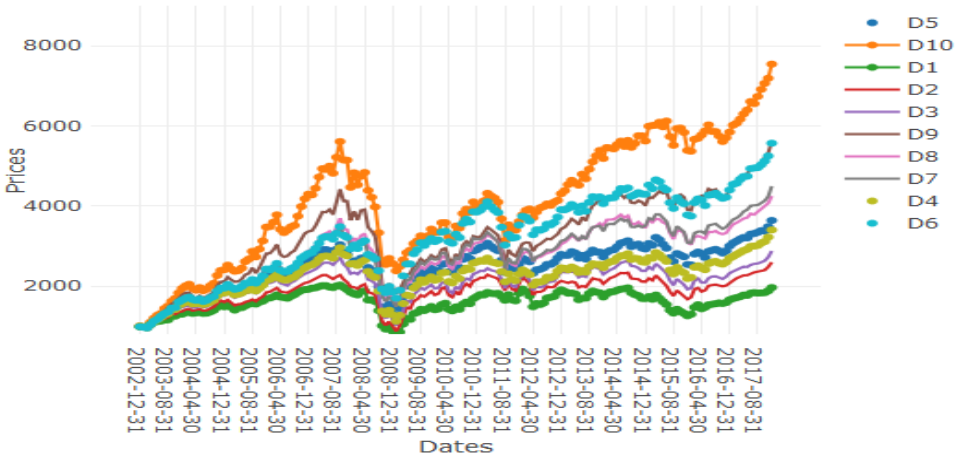


FIGURE 8: **Wealth curve for decile portfolios based Machine Learning model**

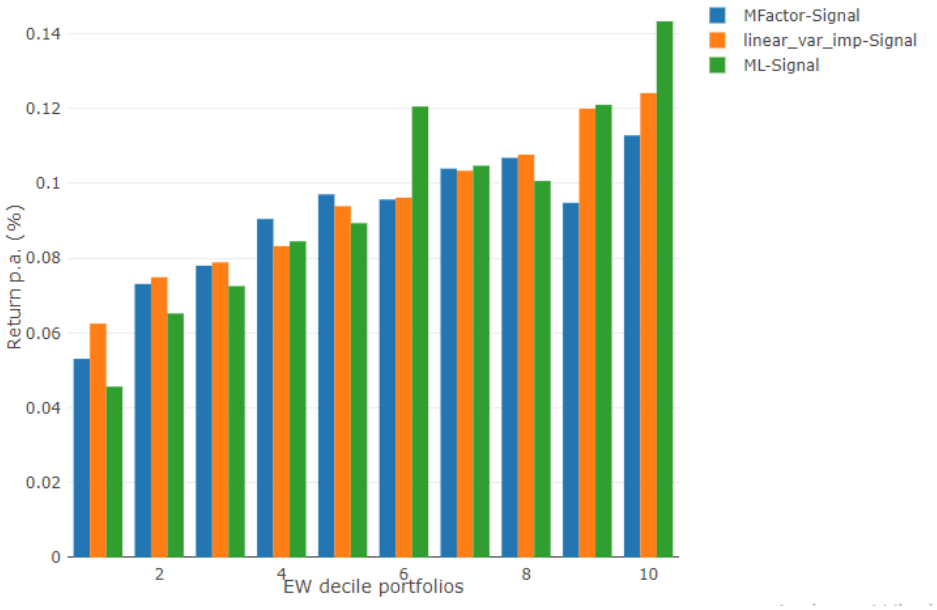


Figures 6, 7 and 8 are wealth curves expressed in dollars for the two benchmarks (multi-factor signal and linear combination of the top 20 features) and the machine learning model using boosted trees classification.
One can note that the three models show a cumulated monotonic performance pattern across the deciles, i.e. the performance of the first decile is lower than that of the second, which lower than that of the third, etc.
The scale of the three graphs is deliberately the same, making the visual comparison much easier. One can see that the dispersion of performance between decile is much clear with the ML model that has been trained to classify sector neutral outperforming and underperforming stocks. The portfolios using the linear combination of the top 20 features is also exhibiting a better cumulated performance monotonic pattern.

5.2. Further evidence of economic gains

In order to further simplify the comparison between our model and the two benchmarks, we plot the annualized return per decile per model in one plot. In Figure 9, one can see that the spread between the average returns of decile 1 and decile 10 is higher for the machine learning model (9.8%), compared to the linear combination of the top 20 features (6%) and to the simple multi-factor portfolio (5.1%).

FIGURE 9: **Annualised performance comparison for each decile of each model.**



The machine learning model benefits here from its tail training were we focused on the top and bottom quintile according to the 1 year forward performance to train the model. Accordingly, and as expected, the ML model yields the worst performance for the lowest decile (D1) and the highest one for the tenth decile (D10).

So far, we have focused our analysis on pure performance, and Table 2 sheds some light on different alternative and complementary metrics of interest. This assesses the robustness of the ML model more deeply.

TABLE 2: **Analytics.** We are comparing the analytics for the top decile (decile 10) portfolio for each model. The two benchmarks on the left and the machine learning model on the right.

	MFactor	**lin_var_imp**	**ML**
Observations	180	180	180
Median monthly return	1.2%	1.5%	1.9%
Annualised return	11.2%	12.4%	14.3%
Annualised volatility	14.7%	19.0%	17.6%
Avg Rank IC (12M)	0.05	0.06	0.11
Avg Rank IC (12M forward vol)	-0.46	0.02	-0.05
Return/Risk	0.76	0.65	0.81
t-stats	2.87	2.39	2.95
Average annual turnover (2 ways)	155%	203%	189%

Analyzing risks measures shows that the multi-factor portfolio has the lowest volatility (14.7%) compared to the linear combination (19%) and the ML approach (17.6%). This result is not surprising: the multi-factor portfolio has 1/5 of the final blended signal that is coming from a low volatility exposure. On top of that, it is well know that certain simple quality associated metrics such as debt to equity, overlap with a low volatility profile.

Regarding the risk-adjusted performance, the ML model generates a Sharpe ratio of 0.81 compared to 0.76 for the multi-factor and 0.65 for the linear combination of top 20 features.

Looking at average rank information coefficient reveals that the ML signal is better to predict the forward 12 months performance. ML signal shows an average IC of 11% compared to 5% for the multi factor and 6% for the linear combination of top 20 variables.
Most interestingly, the average IC numbers for prediction the forward realized volatility show a result of -46% for the multifactor signal. This number should be interpreted as follows: a high level of Multi-Factor signal implies a negative correlation with volatility. Said differently, a high level of Multi-Factor score implies a higher exposure to low volatility stocks. This results is not true with the ML model and the other benchmark.

Finally, the three t-stats of our models are all significant, the highest being the ML one (2.95) compared to 2.87 and 2.39 respectively for the Multi-Factor and the linear combination.

Due to its more dynamic approach, the ML signal generates a higher level of turnover (189%) compared to the Multi-Factor signal (155%). Still, asset rotation is lower than that of the linear combination of the top 20 features.

Results from this section reveal that portfolios based on the ML signals outperformed both benchmarks on a risk-adjusted basis. The ML signal displayed a better IC for one year performance forward and a neutral IC for volatility. A long-short strategy based on the ML signal (long the top decile and short the bottom one) outperforms both benchmarks on a dollar neutral basis.

The non-linear and dynamic approach of the signal based on our machine learning model proved to be more rewarded and more efficient on all metrics (except turnover). This highlights the added value of the boosted tree algorithm, the regularization and the large dataset of features kept to train the model.

Conclusion

In this chapter, we introduce a boosted tree algorithm applied to systematic equity investing. We demonstrate the efficiency of using feature and label engineering. Applying more conditionality and imposing a more causal structure allows a modern quantitative approach to make accurate long-term predictions. This insightful finding contradicts recent criticisms that ML-based approaches were only suitable when predicting very short-term prices movements.

We provide guidance on how to tune, train and test a ML-based model using traditional financial characteristics such as valuation and profitability metrics, but also price momentum, risk estimates, volume and liquidity characteristics. We show that framing the problem is the first priority, and we address it by engineering the features and transformation the labels according to the investment objective.

We find that a naïve equally-weighted portfolio using a boosted tree algorithm with 200 features generates an average outperformance of 3.1%, compared with a simple signal blended multifactor portfolio. Our results also suggest that the ML-based signal is complementary to simple multifactor signals. In a context where the risk of commoditization for equity multifactor portfolios is high, suffering from crowding that could lead to arbitrage of the style equity premia, ML-based signal could constitute an effective remedy for the era post smart beta hangover. The dynamic nature of the signals could constitute a real advantage even in the simplest weighting scheme and implementation process.

Bibliography

Ammann, M., Coqueret, G., & Schade, J. P. (2016). Characteristics-based portfolio choice with leverage constraints. *Journal of Banking & Finance*, *70*, 23-37.

Ang, A. (2014). *Asset management: A systematic approach to factor investing*. Oxford University Press.

Arévalo, R., García, J., Guijarro, F., & Peris, A. (2017). A dynamic trading rule based on filtered flag pattern recognition for stock market price forecasting. *Expert Systems with Applications*, *81*, 177-192.

Ballings, M., Van den Poel, D., Hespeels, N., & Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications*, *42*(20), 7046-7056.

Banz, R. W. (1981). The relationship between return and market value of common stocks. *Journal of Financial Economics*, *9*(1), 3-18.

Bodnar, T., Mazur, S., & Okhrin, Y. (2017). Bayesian estimation of the global minimum variance portfolio. *European Journal of Operational Research*, *256*(1), 292-307.

Brandt, M. W., Santa-Clara, P., & Valkanov, R. (2009). Parametric portfolio policies: Exploiting characteristics in the cross-section of equity returns. *Review of Financial Studies*, *22*(9), 3411-3447.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

Daniel, K., & Titman, S. (1997). Evidence on the characteristics of cross sectional variation in stock returns. *Journal of Finance*, *52*(1), 1-33.

Fama, E. F., & French, K. R. (1992). The cross-section of expected stock returns. *Journal of Finance*, *47*(2), 427-465.

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*(1), 3-56.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119-139.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Annals of Statistics*, *28*(2), 337-407.

Friedman, J. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189-1232.

Friedman, J., Hastie, T., & Tibshirani, R. (2009). *The Elements of Statistical Learning* (2nd Edition). Springer.

Green, J., Hand, J. R., & Zhang, X. F. (2013). The supraview of return predictive signals. *Review of Accounting Studies*, *18*(3), 692-730.

Harvey, C. R., Liu, Y., & Zhu, H. (2016). … and the cross-section of expected returns. *Review of Financial Studies*, *29*(1), 5-68.

Ilmanen, A. (2011). *Expected returns: An investor's guide to harvesting market rewards*. John Wiley & Sons.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, *48*(1), 65-91.

Jegadeesh, N., & Titman, S. (2001). Profitability of momentum strategies: An evaluation of alternative explanations. *Journal of Finance*, *56*(2), 699-720.

Kahn, R. N., & Lemmon, M. (2016). The asset manager's dilemma: How smart beta is disrupting the investment management industry. *Financial Analysts Journal*, *72*(1), 15-20.

Krauss, C., Do, X. A., & Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research*, *259*(2), 689-702.

McLean, R. D., & Pontiff, J. (2016). Does academic research destroy stock return predictability? *Journal of Finance*, *71*(1), 5-32.

Nair, B. B., Kumar, P. S., Sakthivel, N. R., & Vipin, U. (2017). Clustering stock price time series data to generate stock trading recommendations: An empirical study. *Expert Systems with Applications*, *70*, 20-36.

Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, *42*(1), 259-268.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197-227.

Subrahmanyam, A. (2010). The Cross-Section of Expected Stock Returns: What Have We Learnt from the Past Twenty-Five Years of Research? *European Financial Management*, *16*(1), 27-42.

Van Dijk, M. A. (2011). Is size dead? A review of the size effect in equity returns. *Journal of Banking & Finance*, *35*(12), 3263-3274.