Assignment

Machine Learning with Deep Applications in Finance

Imperial College Business School

Andreas Theodoulou
Joshua Ricci
Theodoros Georgantonis
Sanskriti Naik

1. Introduction

In this paper, we aim to identify and evaluate through backtesting a series of trading ideas, constructed through machine learning algorithms for stock predictions, as signals, and portfolio construction methods, as portfolio weighting. The findings are then benchmarked to both a long-only equally weighted portfolio and "market capitalisation" weighted portfolio. The paper introduces the reader to the raw dataset given, discussing additional features included for the implementation of machine learning tools. These mechanism is then outlined with correspondent findings. After portfolio construction, the mechanism is evaluated further for alternative implementations.

2. Data

The data was previously checked to verify that the same amount of information was included for all stocks across years, ensuring a balanced dataset, starting from 2nd February 1999 to 3rd April 2018. Following the exploratory data analysis, upon discovery of a scaling mismatch, the data is standardised for a more accurate investigation. Below, Table 1 reports the statistical findings of this preliminary exploration:

Table 1: Exploratory Description of Original Dataset							
	mean	**std**	**min**	**25%**	**50%**	**75%**	**max**
Mkt_Cap	29712.92	56690.08	17.85	4150.31	10855.34	27504.66	887952.28
P2B	6.87	55.48	0.06	1.69	2.62	4.24	3304.60
Vol_1M	32.08	20.08	1.14	19.24	27.11	38.89	485.030
Div_yield	2.2	2.48	0	0.88	1.81	2.94	90.13
PE_ratio	31.56	145.03	0.37	12.96	17.31	23.01	6626.04
RSI_1M	51.86	8.13	19.7	46.32	51.99	57.47	84.84
D2E	278.69	3311.46	0	35.9	64.38	114.74	130233.33
Prof_growth	19.14	256.69	-1141.14	-3.55	5.92	16.77	21750
Ret_Cap	12.24	13.6	-267.57	6.91	11.34	17.03	927.92
Asset_growth	9.71	29.96	-80.08	-0.86	5.12	12.68	763.89
Prof_Marg	6.97	20.44	-670.05	3.37	7.19	12.02	228.99
F_Return	0.01	0.1	-0.67	-0.04	0.01	0.06	2.26

Table 1: Statistical Summary of Variables

Construction of new features

A total of 11 features are explored, which can be found in Table 1. Many of the features that were already present in the dataset, like market cap, have high persistence through time,

implying that they displayed a high level of autocorrelation through time compared to the dependent variable. Since we want to predict returns, which are noisy, and follow a random walk, we decide to compute additional features, with the intent of eventually allowing the Machine Learning algorithms to explore a vaster variation in the data, and potentially capture better describe the shape of returns. We calculate the 1- month, 3- month and 12- month percentage change of the original features. In the cases that the feature could take the value of zero (e.g. debt-to-equity), we calculate the simple difference from period to period, in order to avoid mathematical errors, such as infinitely large results. Furthermore, the 3-month, 6-month and 12-month momentum features (excluding the last month) are included, together with the 12-month trailing standard deviation of each stock. Put together, our dataset now comprises of a total of 48 parameters.

Standardization and Normalisation

Following the construction of the additional features, the next step is to prepare the data before feeding it into the Machine Learning algorithms. Features with different scaling can cause problems to the ML algorithms due to the level of sensitivity that the attributes exhibit while trying to determine their impact to the model. This is a significant issue, because the algorithms will intrinsically influence the result more due to its larger absolute value, incorrectly prioritising features displaying higher absolute terms than others, ultimately interfering with the accuracy of the results. To overcome this problem, we standardize, or normalise our dataset before conducting any Machine Learning training, so that the values displayed by the features is within a similar range.

We normalise by first cross-sectionally uniformalising the distribution of the features. Then we normalise the previously uniformalised features, as a normal standardisation process would not be sufficient. This procedure is carried exclusively for the features except our dependent variables (returns) as they are already expressed in percentage terms and are therefore already scaled, there is no need no run a normalisation process.

3. Overview of the Machine Learning tools used

Random Forest

Both of our tree models, Random Forest and Boosted Trees are used in a regression setting. By "decision trees", we indicate a number of recursive binary splitting methods, which aim at minimising the number of observation in the wrong class based on the attributes, or features, displayed by the sample. According to the type of tree classification, each method aims at minimising a criteria, such as the Gini coefficient or the Cross Entropy, so that, when a new observation is having to be predicted, the probability of allocating the new observation in the correct class is maximised.

Random forest reduces the variance of the estimator by drawing trees on synthetic copies of the dataset, with observations selected randomly and with replacement from the original dataset, a large number of trees can be created. This way, each tree displays a different yet analogous dataset with the predictors optimising the cut points for each tree at different points, due to the fact that the data fit in each synthetic tree is, consequently, different. The aggregation of these trees for the prediction of a new observation is then utilised to yield a probability, and procedure is called "Random Forest".

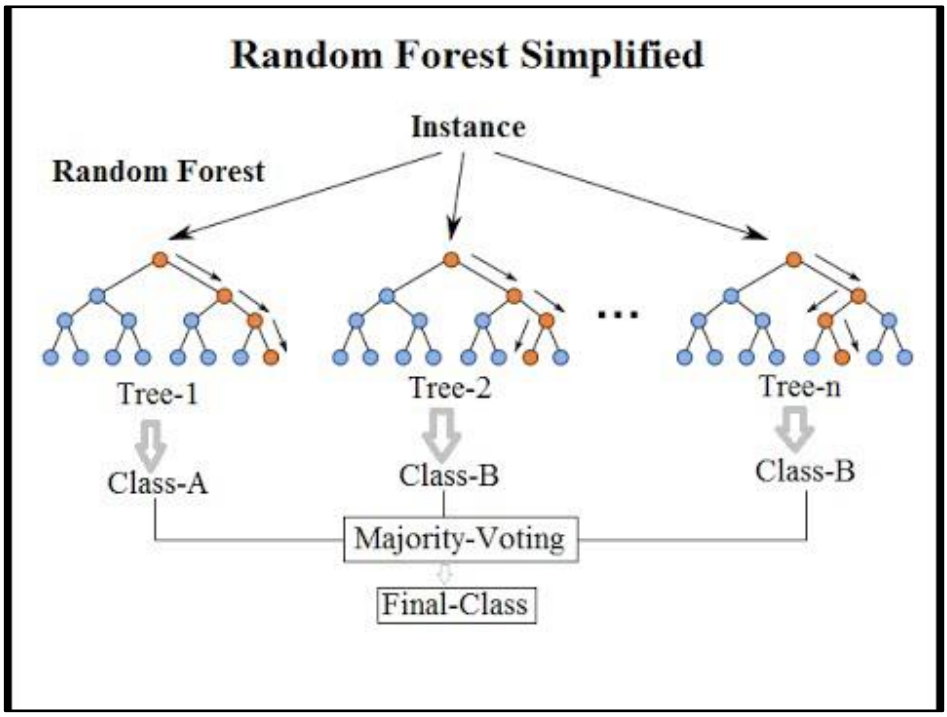

Figure 1: Random forest graph illustration for the classification setting

Boosted Trees (XGBoost)

Similarly to the Random Forest method, “Boosted Trees” are an aggregation of trees utilised for predicting the feature of a parameter. In contrast, Boosted Trees differ from the Random Forest methodology as each new tree is computed by taking into account the dispersion observed in the previously constructed trees and choosing a cut point of the features in order to minimise further the total error in predictability. The model also allows for regularization, on the total number of leaves and on the magnitude of output values and for a learning rate which weights each new tree, and thus tackles one of the major drawbacks of the tree family, namely overfitting.

Neural Network

Neural Networks allow for flexible non-linear functions to be fitted by essentially applying multiple compositions of functions. In practice, a Neural network takes a vector of inputs/features and outputs the prediction. The inputs are fed to a collection of connected units called neurons, where each connection between these neurons can transmit a signal from one neuron to another. The receiving neuron processes the signal, and then signals downstream neurons connected to it. These happens from layer to layer, where at each layer a transformation can be performed to their inputs (the activation function). The signals travel from the input layer to the hidden layers (e.g 3 layers) until the output layer. The graph below gives a brief illustration of the above explanation.

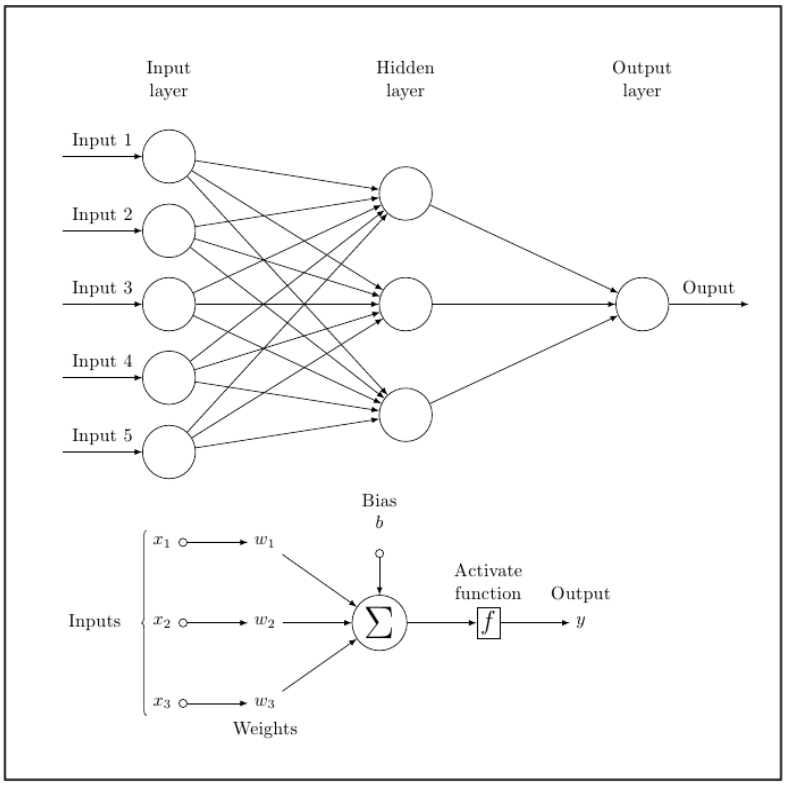

Figure 2: Neural Network graph

Tuning

In simple terms, tuning a combination of parameters, means that we are trying to find the best combination of parameters out of a set of different combinations. This happens by fitting models for each different combination of parameters, and then creating a "validation set" where you test in a theoretically out of sample scenario which combination of parameters would perform best. A metric that usually determines which parameters perform best in the validation test is the mean squared error of predictions and actuals. This helps us determine the effect of varying parameters of the models and determine which to use for prediction in real out of sample scenarios. However, there are numerous other metrics, some of which will be discussed in the implementation method.

4. Implementation – Methodology

Here we discuss our main strategy features and how we have implemented them in our backtest. These can be listed as prediction models, including their tuning, stock selection process and weight optimisation methods.

Prediction Models Specifications and Tuning

As discussed above we choose two models from the family of trees, namely the Random Forest, XGBoost and a 3 Layer Neural Network (3NN) for prediction models. This is because the tree family and Neural Network families have been the ones who have shown to attract more widespread attention and most frequently reported to be top performers in literature and kaggle competitions. We have chosen two type of trees as they are computationally less intensive to tune compared to the Neural Network. For that we will try to compensate on the attention given to the Neural Network by following a bit more closely the "Empirical Asset Pricing Via Machine Learning" paper by Gu, Kelly and Xiu who thoroughly test from 1 to 5 layer Neural Networks for stock prediction and compare them with other popular methods. The outperformance of the 3NN to any other method in the paper has led to us to picking such a specification for our test.

Tuning

For tuning our models we test between a set of 3 parameters for each model, as we find these number to give a precise enough sensitivity and optimisation to the parameters of the model and at the same time a good degree of interpretation. These 3 parameters are chosen based on intuitive effects that change in this parameters will have on the model, explained below, as well as on our research on what's common practice like in the paper "Empirical Asset Pricing Via Machine Learning".

Following common practice, we split the data in a 80-20 frame, allocating a total of 80 months to the test set, where the model is fitted and 20 to the validation set, where the hyperparameters are evaluated and best hyperparameters by our Loss function is chosen. The strategy is then applied on a 12 month rolling window and the cumulative returns is recorded to evaluate the strength of the strategy. The 12 month fitting window was chosen to capture the potential year-by-year changing environment.

For all of our models we have used at least the Hit Ratio as a Loss function. For our Neural Network we have introduced 3 more Loss function which are the Information Coefficient, Mean Squared Error, and the Top/Bottom 20% excess return. We compare the effect of using different Loss functions by comparing the performance of the Neural Network strategy by tuning the parameters on only the Hit Ratio versus tuning on all of the 4 Loss functions, and for each instance putting a ¼ weight to the Neural Network with the tuned parameters that performed best for each metric. Below we define our Loss functions:

- Hit Ratio: Indicates how many times the prediction are in the right direction
- Information Coefficient: The cross-sectional correlation coefficient between forecasts and subsequent returns. The metric comes from the fundamental law of active management introduced by Grinold and Kahn (2000)
- Mean Square Error: The mean squared error of the predicted returns versus the actual returns
- Top/Bottom 20% Excess Return: The difference between the actual mean returns of the top/bottom 20% of the predictions

For each model introduced below we define the parameters tuned, explain our intuition for tuning them and specify the range through which they are tuned in brackets. Our criterion for tuning the parameters are of importance since we believe they have to the sensitivity of the model, the performance and the extent to which they are correlated.

Random Forest

For the random forest we have chosen to tune 3 parameters, maxnodes, ntree and mtry and the rest are left as the default parameters of the "randomforest" package.

- Max Nodes (3,4,5,6)

This tuning method limits the number of terminal nodes the forest can reach. Increasing this attribute allows the leaves of the random forest to display a greater level of classification, although at the expense of overfitting the model. We believe that more than 6 nodes will lead to overfitting.

- Ntree (5,10,20,50)

Controls the total number of trees created while constructing the random forest. This value should be kept above a certain threshold to ensure multiple predictions for each input, in order to produce less noisy predictions.

- Mtry (3, 4, 5, 6)

This attribute controls the number of the features subset that is randomly chosen for each iteration of the random forest. Intuitively, its dependent on the total number of features present in the dataset, therefore the tuning is subjective to the dataset we are dealing with.

This parameters helps to add randomness to the procedure and allows for more features to contribute more equally to the prediction.

XGBoost

For the XGBoost we are tuning max_depth, min_child_weight, nrounds and using the "xgboost" package we leave the rest of the parameters at their default values.

- Max_depth (3, 5, 7, 9)

Controls the maximum number of nodes created throughout from the start point to the furthest node of the tree. The intuition behind the tuning lies within the tradeoff occurring once this parameter is relaxed, as additional nodes cause the tree to become more complex. Nonetheless, there is a level after which an additional node in the tree makes the classification redundant.

- min_child_weight(1, 3, 5)

With this attribute we indicate the minimum weight before an additional node is allowed to be created in the tree. Relaxing this variable allows a greater level of complexity in the classification of the observations, with a similar argument that an excessively low value might lead to problems with overfitting.

- Nrounds (5, 10, 20, 50)

Controls the total number of trees, or iterations, computed throughout the boosted trees process.

3 Layer Neural Network (3NN)

Using the "keras" packaged we developed our neural network. In this case for the parameters that are not tuned we pick the below parameters according to the intuition explained below, and literature researched.

<u>Specification:</u>

- No of units: 32, 16, 8 consequently for layer 1 to 3

We follow the suggested structure in the "Empirical Asset Pricing Via Machine Learning" paper

- Activation function: To be tuned, tanh, sigmoid consequently for layer 1 to 3, and linear for output layer

We use linear for output layer since it matches best the range of value of the variable trying to predict, returns.

- Fixed epochs: 100

We use a big enough number of epochs to allow for more accurate prediction. We tune the parameter that allows us to do early stopping on the number of epochs instead since it's more computationally efficient.

- Fixed baches: 32

Commonly used number, that allows a small enough value to ensure accuracy and not too small to affect computational time

Tuning parameters:

- Optimiser (Adam, Stochastic Gradient Descent)

We understand that the choice of the optimiser used to perform prediction can affect to an extent the outcome. We therefore decide to tune with the above 2 popular methods.

- Activation (Relu, Tanh, Sigmoid)

We tune the activation function for the first layer. We expect this to have a bigger effect on the outcome than any other activation functions of the rest of the hidden layers, as the first layer has the most nodes, and the activation function decides to an extend whether to kill the nodes or not. We use the above 3 popular activation functions to determine their effect on prediction. We aim to improve performance by tuning different activation functions since we expect them to help us capture more variations to the data.

- Min_delta (0.001, 0.001)

Min_delta is a parameter of the apply callback function and determines the minimum improvement required in the loss function between epochs, for the algorithm to stop earlier than the set number of epochs required to run. The patience, which determines how many times this min_delta needs to be reached is set to 7. This parameter controls to an extent how deep the neural network is fitted and therefore helps to control for computational time and overfitting to an extend.

Portfolio Construction and Optimization

After running either of the models described above, we obtain a prediction for the following's month return for each stock of in our investment universe. We call the stage between prediction and actual investment the "portfolio construction" stage, in which the predictions are used to compute the weights. Three differents methods are implemented for the construction of the portfolio weights. Having a score or a return prediction for each stock of your universe is extremely similar with the factor investing approach. Therefore, the common basic idea behind all three methods is that we decide to use only long/short strategies.

- In the first strategy, 50% of the capital is longed in the top X% of the stocks with the highest predictions, while the rest 50% is shorted in the bottom X% of the stocks with the lowest prediction. The capital is allocated in an equally weighted fashion, among these stocks. At this moment we have to mention that we chose for values of 15% or 30% for X%. With a universe of 209 stocks, the top/bottom 15% will produce a long/short portfolio consisting of around 30 stocks, which can be considered fairly diversified.

- The second method comprises of a Minimum Variance optimisation. This procedure follows the exact same process as the method described in the previous point (the stocks in the middle take zero weight), with the only difference that the weight for each stock is chosen by minimizing the overall risk of the portfolio, namely the variance. For this task we used the alabama package. We minimize the objective function, which is $w'\Sigma w$, where w are the stock weights and Σ the variance-covariance matrix. We also constrain the bottom stocks to take weights below zero

and top stocks above zero. Finally, the maximum absolute weight a stock is allowed to be allocated double the one in the equally weighted method.

- In the third and final method, the stock weights are computed by maximizing the Sharpe Ratio. The only difference with the Minimum Variance method is the objective function, w'r/sqrt(w'Σw), where r is the prediction column vector.

We attempt two different ways of estimating the covariance matrix used in the minimum variance and maximum Sharpe ratio optimizations, namely the Historical Covariance estimation, using trailing 60 monthly observations, and the Factor based covariance matrix estimation. In the factor based covariance matrix estimation $Cov[Xt] = B\Sigma B' + \Psi$, where B is the vector of the asset betas, Σ the variance of the market returns (assuming that the market portfolio is the equally weighted portfolio of the 209 stocks given), and Ψ the diagonal matrix of the variances of the residuals. Again, we use the trailing 60 monthly observations to calculate all of the parameters for the calculation of the factor based covariance matrix.

5. Results and Evaluation

In this section we report and evaluate our results for the different variants of strategies we have proposed. We follow the step by step comparison of strategy features we have planned in the implementation phase. Namely we compare our predictive models, our stock selection process and weight optimisation methods. All the results below are reported with an assumption of a 15bps transaction cost on total turnover. The benchmark for our performance is the equally weighted strategy. Our main criteria to evaluate the best strategy is Sharpe ratio and the best strategies are highlighted in yellow.

Models: Random Forest vs XGBoost vs 3 Layer Neural Network

Strategy statistics				
	Boosted Trees 30-30 (EW)	**Random Forest 30-30 (EW)**	**3NN 30-30 (EW)**	**EW strategy**
Annualised Return	5.64%	5.65%	5.08%	7.28%
Annualised Std	13.97%	7.14%	7.04%	20.93%
Sharpe ratio	0.40	0.79	0.72	0.35
Max Drawdown	23.69%	11.49%	11.63%	52.39%
Mar Ratio	0.24	0.49	0.44	0.14
Mean Turnover	141.43%	90.52%	81.98%	0.76%
Historical Var (5%)	-4.47%	-2.57%	-1.65%	-10.46%
Parametric Var (5%)	-6.10%	-2.91%	-2.91%	-9.17%
% Positive Months	57.25%	58.02%	55.73%	60.31%
Downside Deviation	7.71%	3.69%	3.13%	13.79%
Sortino Ratio	0.73	1.53	1.62	0.53

Table 2: Comparison of Statistics Across Strategies

All models are compared for a Long/Short (LS) top/bottom 30% for Equally Weighted (EW) portfolio weights.

Stock selection process: Long/Short 30% Vs Long/Short 15%

This two cases of L/S top/bottom 30% and L/S top/bottom 15% are compared for the equally weighted portfolio picked based on a random forest prediction, since random forest has proved to be the best prediction among the 3 tested models.

Strategy statistics		
	Random Forest 30-30 (EW)	**Random Forest 15-15 (EW)**
Annualised Return	5.65%	9.53%
Annualised Std	7.14%	10.89%
Sharpe ratio	0.79	0.88
Max Drawdown	11.49%	15.79%
Mar Ratio	0.49	0.60
Mean Turnover	90.52%	123.96%
Historical Var (5%)	-2.57%	-3.44%
Parametric Var (5%)	-2.91%	-4.36%
% Positive Months	58.02%	61.07%
Downside Deviation	3.69%	5.17%
Sortino Ratio	1.53	1.84

Table 3: Random Forest Strategy Statitsics

We can see that the 15% L/S strategy outperforms the 30% one. This could be expected to an extent since 30%L/S trades 60% of the universe, which seems to be a high percentage to achieve abnormal returns. On the other 15% as explained earlier seems a reasonable enough percentage for our universe to allow for diversification.

Weight Optimisation method: EW vs Min Variance vs Max Sharpe

a) The below weight optimisation methods are performed on a Random forest based strategy with a 15% L/S. Here, we try to test whether our weight optimisation method can further enhance the performance of our strategy. Note that the covariance matrix for the optimisation of Min Variance and Max Sharpe is calculated through the Historical method.

Strategy statistics			
	EW L/S 15%	MinVar L/S 15%	MaxSharpe L/S 15%
Annualised Return	9.53%	6.74%	8.20%
Annualised Std	10.89%	6.16%	7.04%
Sharpe ratio	0.88	1.09	1.16
Max Drawdown	15.79%	7.63%	8.91%
Mar Ratio	0.60	0.88	0.92
Mean Turnover	123.96%	152.65%	155.50%
Historical Var (5%)	-3.44%	-1.83%	-2.22%
Parametric Var (5%)	-4.36%	-2.36%	-2.67%
% Positive Months	61.07%	59.54%	60.31%
Downside Deviation	5.17%	2.75%	2.99%
Sortino Ratio	1.84	2.45	2.74

Table 4: Comparison of Weighting Schemes on Random Forest Strategy

b) Here we test if we find any difference when we alter the method of estimating the covariance matrix used in the Max Sharpe ratio calculations (and Min Variance), from the Historical to the Factor based estimation method. We find that this has a negligible effect.

Covariance estimation method		
	Historical	Factor based
Annualised Return	8.20%	8.20%
Annualised Std	7.04%	7.03%
Sharpe ratio	1.16	1.17
Max Drawdown	8.34%	9.74%
Mar Ratio	0.92	0.84
Mean Turnover	155.50%	148.42%
Historical Var (5%)	-2.22%	-2.35%
Parametric Var (5%)	-2.67%	-2.66%
% Positive Months	60.31%	65.65%
Downside Deviation	2.99%	3.45%
Sortino Ratio	2.74	2.38

Table 5: Performance of Covariance Estimation Method

4 Loss Functions vs 1 Loss Functions

As a final attempt to challenge the dominance of the Random Forest model we use another variation of a NN3 based (2nd best model) Strategy. We keep the L/S 15% stock selection process with the Min Variance optimisation for both methods. Here we use 4 Loss functions in our validation set and for each loss function we put ¼ weight to the prediction of each of the best parameter-tuned model, according to the respective loss function. We can see that

introducing 4 loss function does improve slightly the results of the NN3 strategy, for example the sharpe ratio from 0.96 to 1.04. However, it still slightly underperforms compared to the best version of the Random Forest (L/S 15%, Max Sharpe) and 2nd best (L/S 15%, Min Variance).

NN3 4 Loss functions vs 1 loss functions (both with min variance, 15% L/S)		
	1 Loss	**4 Loss**
Annualised Return	6.26%	6.86%
Annualised Std	6.50%	6.62%
Sharpe ratio	0.96	1.04
Max Drawdown	8.34%	11.82%
Mar Ratio	0.75	0.58
Mean Turnover	135.18%	144.82%
Historical Var (5%)	-1.74%	-1.87%
Parametric Var (5%)	-2.56%	-2.57%
% Positive Months	61.83%	64.89%
Downside Deviation	2.65%	3.43%
Sortino Ratio	2.36	2.00

Table 6: Impact of the Use of Loss Functions on Neural Network Strategy

6. Conclusions

Firstly, after the various strategies we have tried it seemed to be persistent enough that the random forest outperforms the XGBoost and the Neural Network models for the parameters we have test. Secondly, it seems that for the 15% L/S strategy outperforms the 30% one which has been consistent with all of the models we have tested, and we have reported the results of the Random Forest case. Lastly, in terms of portfolio weight optimization the max Sharpe ratio outperforms to a satisfactory level the minimum variance and equally weighted portfolio. As a result we conclude that for the cases tests and the specific hyperparameters the Random forest with a L/S 15% portfolio and with max Sharpe ratio optimasion, is the best performing strategy.

We further outline some further improvements that we believe could be valuable as addition to the work we performed. Firstly, to be more confident for our results, the deflated sharpe ratio could be analysed to test whether the performance of our final Random Forest strategy, is significant enough statistically. Also, to further improve our strategy we could have potentially test an ensemble technique combining the top 2 or all 3 of the strategies. This could help us capture more variation in the data, consequently more accurate returns predictions and thus potentially better strategy performance.

7. Appendix

The average hit ratio over the 12 tunings made over the backtesting period is reported for the different variations of the top performing model. We report the top 7 average hit ratio combination of parameters for each model, except for Neural Networks where we report all the combinations since the number of combinations is small.

Table 1: XGBoost Tuning

XGBoost Tuning			
max_depth	**nrounds**	**min_child_weight**	**avg_hit**
3,5,7,9	**5,10,15,20,50**	**1,3,5**	**>= 0.568**
7	5	5	0.5689
9	5	3	0.5686
7	5	3	0.5685
9	5	1	0.5683
3	5	5	0.5681
3	5	1	0.5681
3	5	3	0.5681

Table 2: Random Forest Tuning

Random Forest			
maxnodes	**ntree**	**mtry**	**avg_hit**
3,4,5,6	5,10,20,50	3,4,5,6	
3	20	5	0.6066
3	5	5	0.6063
3	50	4	0.6062
3	5	4	0.6061
3	50	5	0.6059
4	10	6	0.6059
3	20	4	0.6057

Table 3: Neural Network Tuning

NN3 Tuning			
min_delta	**activation**	**optimizer**	**hit**
1.00E-03	adam	relu	0.5511
1.00E-04	adam	relu	0.5329
1.00E-03	adam	tanh	0.5100
1.00E-04	adam	tanh	0.5373
1.00E-03	adam	sigmoid	0.5311
1.00E-04	adam	sigmoid	0.5122
1.00E-03	sgd	relu	0.5237
1.00E-04	sgd	relu	0.5395
1.00E-03	sgd	tanh	0.5328
1.00E-04	sgd	tanh	0.5355
1.00E-03	sgd	sigmoid	0.5264
1.00E-04	sgd	sigmoid	0.5484

Table 4: Neural Network tuning with 4 loss function (average)

Neural Network 3 with 4 lost functions						
min_delta	**activation**	**optimizer**	**Hit**	**IC**	**TB20**	**MSE**
1.00E-03	relu	adam	0.5196	6.44E-02	2.79E-03	0.007959
1.00E-04	relu	adam	0.5120	0.079029	3.48E-03	0.008074
1.00E-03	tanh	adam	0.5117	0.076149	3.96E-03	0.007964
1.00E-04	tanh	adam	0.5169	0.064201	3.11E-03	0.00819
1.00E-03	sigmoid	adam	0.5236	0.112086	4.68E-03	0.007783
1.00E-04	sigmoid	adam	0.5195	0.100459	3.32E-03	0.007951
1.00E-03	relu	sgd	0.5105	0.037132	1.80E-03	0.008222
1.00E-04	relu	sgd	0.5149	0.05553	2.63E-03	0.007988
1.00E-03	tanh	sgd	0.5008	0.020473	1.22E-03	0.008389
1.00E-04	tanh	sgd	0.5194	0.056129	2.58E-03	0.007986
1.00E-03	sigmoid	sgd	0.5023	0.002695	3.61E-04	0.008076
1.00E-04	sigmoid	sgd	0.5064	0.025799	1.50E-03	0.007959