Combining Stacked Ensembling & PCA for Log-Optimal Bet Sizes

UCD School of Computer Science

University College Dublin, Belfield, Dublin 4, Ireland

Abstract. This study investigates how a stacked ensemble methodology, combined with Principal Component Analysis(PCA) and a custom objective function, can be combined in to predict bets that would achieve log-optimal growth on the FTSE100. The model is evaluated on 156 days of testing data, showing high accuracy and F1 scores. A trading simulation is also run over this period with the investment strategy generated by the models outperforming a buy and hold strategy over the same period on both cumulative return and Sharpe Ratio.

Keywords: Deep Learning · Time Series · Ensembling

1 Introduction

The challenge of accurately predicting Time Series data, such as equity prices, bond prices and volatility levels is one of the most difficult in the finance industry. The Efficient Market Hypothesis(EMH) suggests it is not possible, however. Over the past few years, there has been a lot of research on how machine learning techniques can be applied to outperform classic time series forecasting techniques such as ARIMA modelling. The constant battle between investors and speculators seeking higher yields means that new methods and models are constantly being developed and fine tuned in order to gain an edge in the market. Early attempts to use machine learning for forecasting financial time series started with some simple Artificial Neural Networks(ANNs) around 1964. Since then there has been an abundance of focus both in academia and in industry on improving the forecasting ability of these models. Knowing where the market will be tomorrow gives a strong financial incentive to innovate, after all. This study takes a slightly different approach to most. Instead of trying to predict the returns themselves, the intention is to find bet sizes that would achieve log-optimal growth. A combination of deep learning methods are used and compared in a stacked ensemble like approach in order to achieve this.

One of the main difficulties when modelling Financial Time Series data is the low signal-to-noise ratio. This makes it difficult for models to discern random fluctuations from meaningful signals that provide information on where the asset's price will go next. Many noise removal techniques have been tried over the years. For this study, a combination of truncating the returns between a certain threshold and performing a statistical transformation in the form of PCA are used in order to filter out noise while maximizing the amount of information passed to the models.

2 Related Work

2.1 Ensemble Methods

Meta Labelling Meta-labelling [2] is where sub-learners predicts the side of the bet, with a 1 representing a long position and a -1 representing a short position, similar to a binary classification task. These predictions, along with their corresponding probabilities are then fed into a secondary model(meta learner) as features. The Secondary model determines whether or not the bet should be plaved, which can be either 0 if the meta learner determines the sub learner to be incorrect, or 1 if the sub learner deems the sub learners' prediction to be correct.

The advantage of this methodology is that a bet does not always have to be placed: If the meta learner determines the sub learner was wrong, it does not take the opposite side of the trade, instead it takes up no position. This can be interpreted as a weak signal being present in the data at that point in time. Another advantage of this method is the output probabilities help give a confidence measure of the sub-learners predictions. When we do regression of (transformed) prices, these probabilities are not available and hence the meta learner cannot know how confident a model's prediction is.

Bagging and Boosting Boosting[9] creates many weak learners and reweighs samples that are difficult to classify so the learners focus more on their correct prediction. There are multiple rounds of boosting in order to allow for this methodology to be effective. Boosting effectively reduces the bias of predictions and many variants of boosting, such as XGBoost [4] and LightGBM [12] which have been shown some good results in financial time series modelling, for example in[10].

Bagging(Bootstrap aggregating)[13] is another ensemble method that has been used in financial time series modelling to increase prediction accuracy. Bagging relies on the bootstrap methodology which generates subsets of the training data and fits multiple models on this data. The final predictions are then based on a weighted average or a majority vote of all the models. Bagging helps to reduce over-fitting.

2.2 Hybrid Methods

Hybrid methods, which combine statistical and deep learning methods have shown impressive results in time series forecasting problems. By combining these two methodologies, patterns in financial time series data can be more effectively captured due to the cancellation of errors and noise as a result of averaging. Smyl [14] used Exponential Smoothing and a LSTM, trained using the same gradient descent method to outperform deep learning and statistical methods in a Kaggle competition (M5) on time series forecasting.

2.3 Deep Learning Paradigms

The M5 competition hosted on kaggle.com has acted as an incubator for new approaches and techniques to time series forecasting. N-BEATS has shown to be . HIVE-COTE is an example of an ensemble based approach to time series modelling, being described as the state of the art algorithm for time series classification. Recent advances in time series forecasting

2.4 Datasets

Benchmark datasets like Mackey-Glass have been used in the past to compare the ability of time series models. Financial data is inherently noisy - it has a very low signal-to-noise ratio. This makes it difficult to train any type of algorithm on the data, be that The dataset used is daily Closing Price and Volume data from the FTSE 100 index. This is a well known index with many reliable data sources. The data goes back as far as Financial data is highly nonstationary,

3 Model Architecture

3.1 Overview

Stacking has been shown in the past to reduce bias when used for many machine learning tasks, while Deep Learning methods such as RNNs have proven very successful in the past in modelling sequential data such as Natural Language and speech recognition problems[**?**]. This work aims to combine both of these methods on Financial Time Series data, along with Principal Component Analysis(PCA)[5].

The Deep Neural Networks used in this study use the ADAM algorithm which is a Gradient-Descent type algorithm. If the ranges of features fed into Gradient-Descent type algorithms are different, the step sizes will be different for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, the data was scaled before feeding it to the models. Having features on a similar scale will help the gradient descent converge more quickly towards the minima, making the training faster. Normalizing the data also prevents the optimization from getting stuck in local optima as it gives a better error surface shape. Bayes optimization, which is also used in this study to find model hyper-parameters, can be done more conveniently with scaled data.

Seven lags of transformed price return data and one lag of volume data are normalized using min-max scaling and fed into a sub learner, a Multi-Layered Perceptron. The sub-learner hyper-parameters are found using purged Cross-fold Validation. The sub-learner one-step predictions along with lag price return and volume return data are transformed using PCA and normalized using min-max scaling before being fed into the meta-learner.

The meta learner then predicts the optimal investment size to achieve log-optimal growth, based on the 'Merton Ratio' which is μ_i/σ_i. If σ_i is constant then the best size is proportional to μ_i, the forecast return for day i.

The Meta Learner generates one step predictions for each input sample, and the output of the meta learner is a vector of each of these one step predictions, concatenated together.

Three different Meta-Learners are considered: An MLP, Vanilla RNN and an LSTM.

3.2 Sub-Learner

Deep Multi-Layered Perceptron A Deep Multi-Layered Perceptron with 4 hidden layers is used as the base learner. Each hidden layer gets passed through a ReLU activation function. For each of the 10 splits for generating predictions to be fed into the Meta Learner, a different model is fit. Each of these models is found by using Purged Cross fold validation with a gap of 7 for each split. For the portion of the data to be used as testing and validation data for the Meta Learner, Purged Cross fold validation is used again on the entire training dataset and the best model found is used to predict one step prediction values for these datasets. Figure 1 shows the architecture of a MLP.

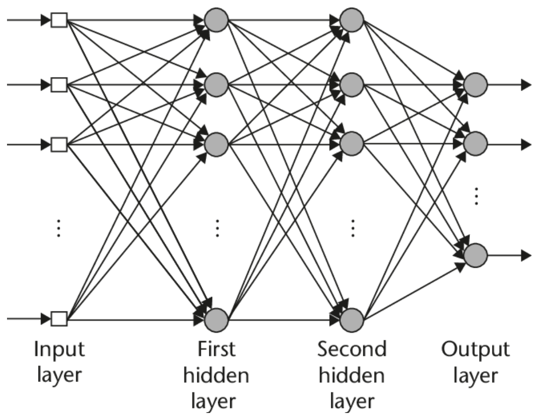

Fig. 1. MLP Architecture

3.3 Meta-Learners

Recurrent Neural Network Recurrent Neural Network(RNNs) are Neural Networks designed for sequential data which typically comes from areas such as speech recognition and translation, natural language processing and time series data. Classical NNs such as the MLP are capable of handling sequential data, but have some limitations, such as the inability to handle sequences of variable length and to detect time invariant patterns in the data.[6] RNNs make use of recurrent connections which connect the hidden units in the network back to themselves with a time delay. RNNs main pitfall is their training difficulty. A long sequence of data modelled by an RNN gives rise to a strong possibility of vanishing or exploding gradients[7], making it impossible to build an accurate model. Various alterations to RNNs, such as LSTM and GRU have been found useful in avoiding this issue. Figure 2 shows the architecture of an RNN cell.

If the the input value of the t^{th} observation is $x_t = (x_{t,1}, \dots, x_{t,n})$ then an RNN can be described mathematically as

$h_t = \tanh(\mathrm{U}x_t + \mathrm{W}h_{t-1} + \mathrm{b})$

$o_t = \tanh(\mathrm{V}h_t + \mathrm{c})$

Where h_t is the hidden state at time t and o_t is the output of the model b and c are bias terms while U, W, V are input-to-hidden, hidden-to-hidden and hidden-to-output weight matrices respectively, which are calculated during training.

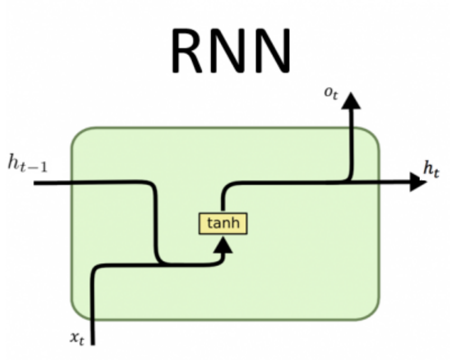

Fig. 2. RNN Cell Architecture

Long Short Term Memory Long short term memory(LSTM)[8] is a modified version of an RNN. It can remember both short and long term values and is capable of learning order dependence in sequence prediction problems. LSTMs have a forget gate, an input gate and an output gate. These gates are fully connected layers with weights that can be trained using well known back-propagation techniques.

This enables the LSTM to regulate the flow of information as the gates decide which data is important and which data should be thrown away. By doing so, it can pass relevant information down the long chain of sequences to make predictions. LSTMs use back propagation via the ADAM algorithm in order to optimize the parameters of these gates. LSTMs avoid the vanishing/exploding gradients problem experienced by RNNs.

Figure 3 shows the architecture of an LSTM cell.

If the the input value of the t^{th} observation is $x_t = (x_{t,1}, \dots, x_{t,n})$ then an LSTM can be described mathematically as

$i_t = \sigma\ (U_i x_t + \mathrm{W} h_{t-1} + b_i)$
$\tilde{C_t} = \tanh(U_c x_t + W_c h_{t-1} + b_c)$
$f_t = \sigma\ (U_f x_t + W_f h_{t-1} + b_f)$
$C_t = i_t {*} \tilde{C_t} + f_t {*} C_{t-1}$
$o_t = \sigma(U_0 x_t + W_0 h_{t-1} + V_0 C_t + b_0)$
$h_t = o_t {*} \tanh(C_t)$

where σ is the sigmoid activation function, U_i,W_i, U_c ,W_c are the weight matrices, b_i and b_c are the bias terms, h_t is the hidden state at time t and o_t is the output of the model.

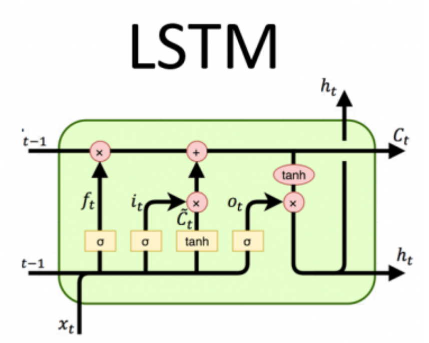

Fig. 3. LSTM Cell Architecture

4 Methods

4.1 Creating the Dataset

Raw Data The raw data contains Closing Price(Figure 4) and Volume(Figure 5) data from the FTSE100 index, obtained from Yahoo! Finance Python API and covers a time period from 1985 until 2021.

Feature Engineering The first step was to identify any NaNs in the dataset, and remove them.

As can be seen from the plots, the raw data is highly non stationary, especially price data, as the mean of the series varies throughout time. The raw data was transformed in order to make the series stationary by taking first order differences. Regression models require the data to be stationary because the mean of a stationary time series is constant throughout time. We are attempting to model bet sizes that are proportional to the mean over multiple time periods, so if this mean varies the Machine Learning algorithm used will not be able to account for this, leading to an unsuccessful trading strategy.

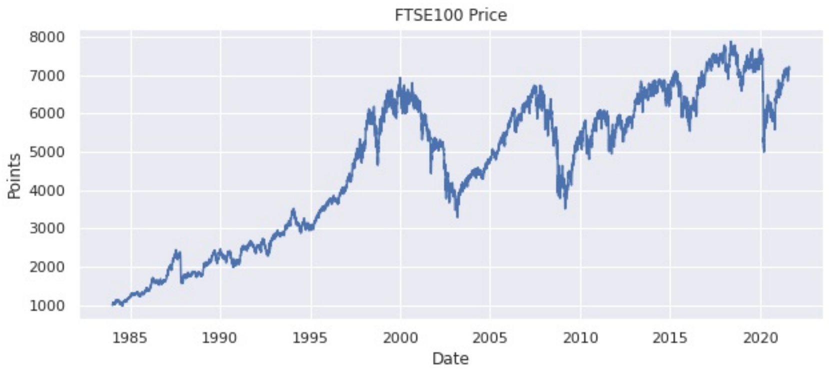

Fig. 4. FTSE100 Historical Price

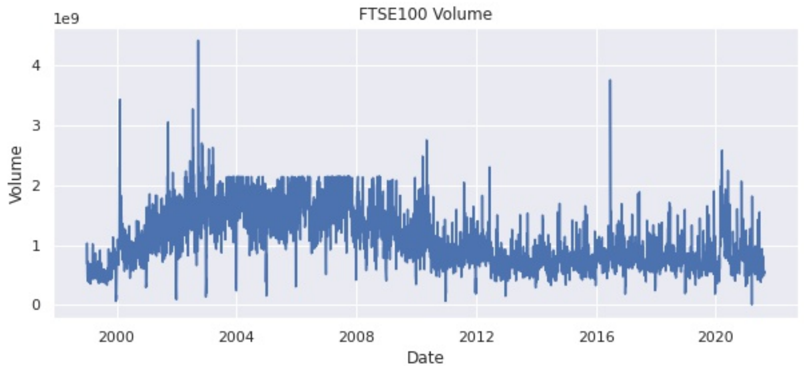

Fig. 5. FTSE100 Historical Volume

A natural logarithm transformation was also applied, as stock returns have been found to closer resemble a log-normal distribution rather than a normal distribution.

In order to reduce the amount of noise fed into the models, the log-returns are then truncated between -0.03 and 0.03 by dividing by 0.03, taking the hyperbolic tangent and finally multiplying by 0.03, such that

$y_t = 0.03*\tanh(\log(\frac{x_t}{x_{t-1}}))$

The resulting time series can be seen in Figure 6 and its distribution can be seen in Figure 7.

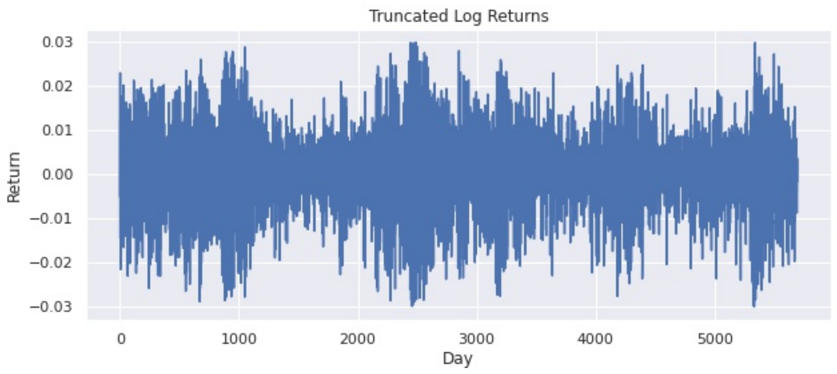

Fig. 6. FTSE100 Truncated Log Returns

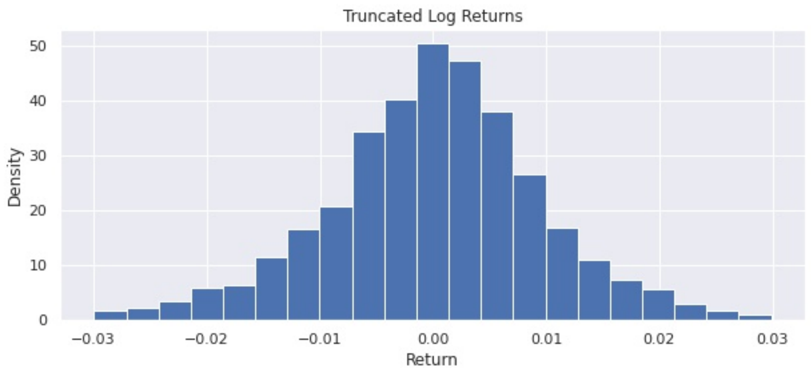

Fig. 7. Histogram of FTSE100 Truncated Log Returns

A similar methodology was applied to volume data. Initially the natural log of Volume was considered as a feature input but some decision tree feature importance analysis indicated that it was not a useful feature. In order to overcome this, I took the first order difference of Volume data, took the natural logarithm, applied hyperbolic tangent and multiplied by 0.03 so that it would be on the same scale as the price return data. The resulting data can be seen in Figure 8

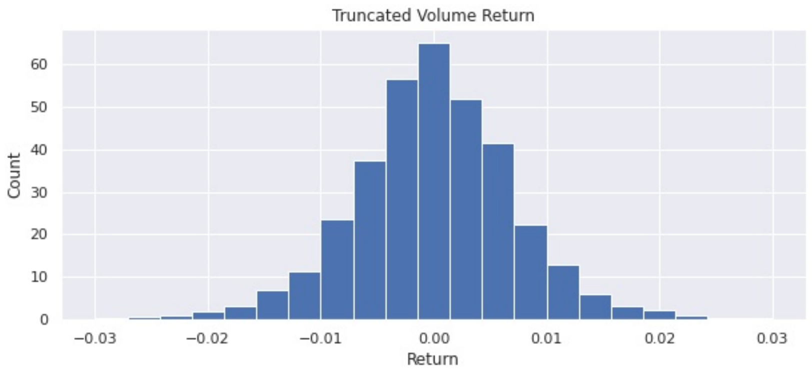

Fig. 8. FTSE100 Truncated Volume Returns

The dataset provided only contains volume data back as far as the year 2000, hence all data from before that period is not used in any of the model building or evaluation.

Summary Statistics The summary statistics for the transformed price returns can be seen in Table 1, while Figure 9 shows a balanced split of the data between positive(Blue) and negative returns(Orange).

Table 1. Summary Statistics for Transformed Price Returns

Mean	St.Dev	Min	25%	50%	75%	max
0.000110	0.009503	-0.029	-0.005404	0.00422	0.005856	0.029

From Time Series to Supervised Dataset In order to turn the dataset into a supervised machine learning problem, 8 lags of the transformed price return, as well as one lag (lag1) of transformed volume were generated. The first price lag, (Lag0) was then separated from this dataset to be used as the target(y) variable.

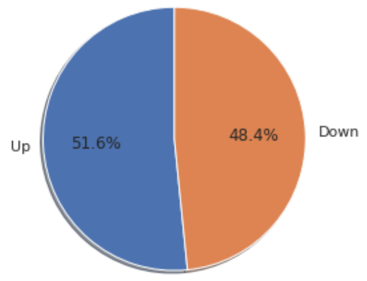

Fig. 9. Split of Training Dataset

4.2 Stacking Framework

Stacking[3] is an ensemble algorithm which combines the predictions of lower level learning algorithms into a higher level meta learner, with the intention of greater prediction performance.

In this study, a modification is made to this whereby the the original dataset is combined with predictions from lower level learner. Figure 10 depicts the process in detail.

This stacked dataset is transformed, first using PCA and secondly with MinMaxScaling before being fed into the meta-learner.

5 Experimental Setup and Evaluation

5.1 Partitioning the Dataset

9:1 was the Training:Test split for training the sub learners. This resulted in 570 samples for testing and 5123 for training.

For training the Meta Learner, due to the nature of TimeSeriesSplit, only 77% of the data used to train the sub learner was used for training the meta learner. This is because the predictions for this first 23% could not be generated out of sample, due to the sequential ordering of TimeSeriesSplit. Using in sample predictions in stacking can lead to biased predictions and would lead to overfitting if the predictions were to be included in the data fed into the Meta Learner.

A validation dataset was used to optimize the model hyper-parameters. This dataset was taken from the stacked test set rather than the stacked train set. Taking from the train set would mean are tuning parameters on the same data the model is fit on, which would lead to overfitting and poor generalization ability.

Seven samples from the start of the validation set and seven samples from the start of the test set were removed before any hyper-parameter tuning/ evaluation were performed. This was to avoid data leakage and to help avoid overfitting.

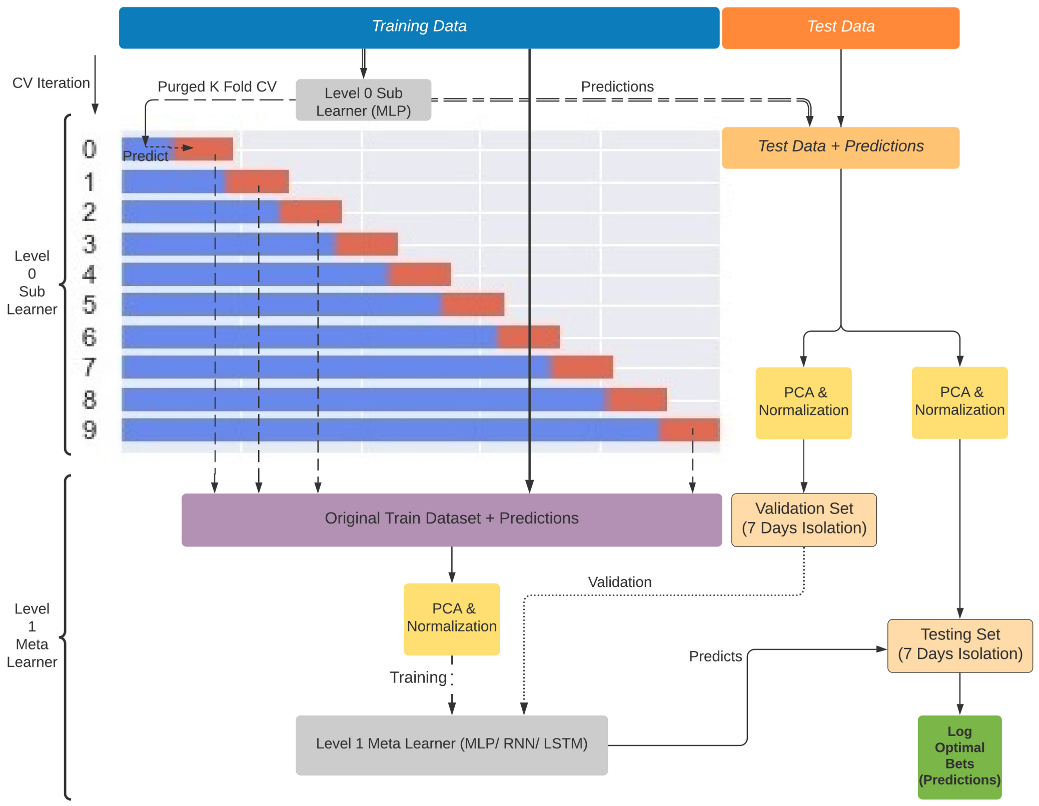

Fig. 10. Altered Stacking Methodology

The amount 7 was chosen as this is the number of time lags that are fed into the Meta Learners.

Hence the split of the dataset for the Meta Learners was: 3944 Train Samples, 309 Validation Samples and 149 Test samples. Or 90% Train, 7% Validation and 3% for testing.

5.2 Principal Component Analysis

PCA was applied to the data before being fed into the meta learner. PCA transforms the features so they are orthogonal to eachother, which is a desirable characteristic of a dataset fed into a neural network as typically neural networks will waste time on learning the correlation if the features are correlated. This approach ensures the learning does not focus on autocorrelation, due to the nature of PCA always making orthogonal features.

5.3 Evaluation Metrics

The modelling setup is similar to typical regression tasks where we try to predict E(y) using a model m, such that m(X) = E(y). However, the purpose of this research is not predict E(y) but rather the optimal wealth fraction to invest for a given yi. This value should be proportional to yi, but it means that typical Regression metrics such as RMSE (Root-Mean Square Error), MAE(Mean Absolute Error) and MAPE(Mean Average Percentage Error) are not relevant for evaluating the performance of the meta learner, as we are not trying to predict returns themselves, rather the optimal bet size.

A very important element when designing a ML strategy for trading is that you get the direction right. Hence we use evaluation metrics more associated with classification tasks; accuracy and F1-score. Cumulative return and Sharpe Ratio are used as evaluation metrics for the trading simulation.

Loss Function The loss function used is of the form

$$(m(x)y - 1)^2$$

where m(x) is the output/prediction of the meta learner.

The predictions are bet sizes. Ideally the bet size is the Merton Ratio, similar to the Sharpe Ratio except with variance in the denominator instead of standard deviation. Generally the Merton Ratio is the bet size to achieve optimal growth. If the variance is constant then bet size is proportional to the expected return for day i.

The evaluation metrics are analysed using the following formulae where TP = True Positive, TP = False Positive, TN = True Negative and FN = False Negative.

Precision $\frac{TP}{(TP+FP)}$

Recall $\frac{TP}{(TP+FN)}$

Accuracy $\frac{TP+TN}{(TP+FP+TN+FN)}$

F1-Score $\frac{2*Precision*Recall}{(Precision+Recall)}$

Total Return Σ(Bets*Actual Returns)

Sharpe Ratio The Sharpe Ratio[11] is a measure of the risk to return trade-off of an investment strategy. A very good strategy will have a Sharpe ratio greater than one. The ratio is the average return earned in excess of the risk-free rate per unit of volatility. Volatility is a measure of the price fluctuations of a investment strategy, with the standard deviation of the return of the investment strategy taken to be the measure of volatility for the purpose of this research. A risk free rate of 0.0007% is used for this study.

Subtracting the risk-free rate from the mean return allows an investor to better isolate the profits associated with risk-taking activities. The risk-free rate of return is the return on an investment with zero risk, meaning it's the return investors could expect for taking no risk. The yield for a U.S. Treasury bond, for example, could be used as the risk-free rate.

Generally, the greater the value of the Sharpe ratio, the more attractive the risk-adjusted return.

5.4 Hyper-parameter Tuning

Hyper-parameters are the variables of a network that effect the network architecture and performance. The number of neurons in each layer, the number of hidden layers, the optimization algorithm to use, the learning rate, decay rate, momentum values, number of epochs, (mini) batch size are the hyper-parameters of the network.

K-fold cross validation is used in modelling many types of data. It has been shown to help improve model performance, such as in [10]. There are many financial papers that use K-Fold CV with the claim their model performs well. However, it is very unlikely that these results are meaningful. K-fold CV does not work well with financial data is because observations cannot be assumed to be drawn from an IID(Independent and Identically distributed) process. Secondly, the out of sample (test dataset) is used multiple times while building the model - this results in bias in both testing and in parameter selection.

The problem of data leakage occurs when observations from the training set also appear in the test set. Stock returns have been shown to follow serial correlation, and because of that, $X_t \approx X_{t+1}$ Hence having t and t+1 in different sets causes data leakage, which leads to results with over inflated accuracy measures.

Purged K Fold cross validation [2] avoids data leakage and bias by adding a gap (of size 7 in this case) between the training and testing folds.

Bayesian Optimization Finding a globally optimal solution for black-box models such as MLPs and RNNs using exhaustive hyperparameter tuning methods such as GridSearchCV would be extremely computationally and time intensive and were infeasible for this study. Hence, the Meta learner hyper-parameters were optimized using Bayesian Optimization[1]. Bayesian optimization is based on Bayes Theorem, with the optimization of hyper-parameters being guided by a probabilistic model of the objective function called a surrogate function. The surrogate function can be searched efficiently with an acquisition function before candidate samples are chosen for evaluation on the real objective function. Keras Tuner implementation of Bayesian Optimization is used in this study to evaluate the model parameters.

Unlike exhaustive search, where each combination of hyperparameter is run and evaluated, Bayesian optimization takes as input a maximum number of trials to run. Each trial can be run multiple times in order to reduce the effect of random initialization on the trained models.

The model hyper-parameter set is then selected based on those that gave the lowest loss on the validation dataset. For this study, for each meta-learner, 50 trials were run, with 2 executions per trial in order to reduce the effect of random initialization on the trained models. Early stopping was also used as a regularization technique to prevent over-fitting.

Table 2. Hyper-parameters

Model	Hyper-Parameter	Search Space	Final
Sub Learner MLP	Units per Layer	16,32,64,128	32
	Num Layers	3,4,5	4
	Learning Rate	0.0001,0.001,0.01	0.001
	Max Epochs	1000	1000
	Batch Size	50,100,200	100
Meta-Learner MLP	Layer1 Units	32 - 128	32
	Layer1 Dropout	0 - 0.2	0.02
	Layer2 Units	8 - 96	96
	Layer3 Units	8 - 96	80
	Layer4 Units	8 - 96	96
	Learning Rate	1e-4 - 1e-2	0.0003
	Max Epochs	300	300
	Batch Size	32,64,128	64
Meta-Learner RNN	Units per Layer	4-80	4
	Num Layers	1,2	1
	Learning Rate	1e-5 - 1e-1	0.012
	Max Epochs	20-100	100
	Batch Size	16-128	80
Meta-Learner LSTM	Units per Layer	4-80	4
	Num Layers	1	1
	Learning Rate	1e-5 - 0.1	0.05
	Max Epochs	20-200	100
	Batch Size	16-128	64

6 Results

6.1 Meta Learners

Table 3. In Sample (Train Data) Performance Metrics

Description	(SL)MLP	(ML)MLP	(ML)RNN	(ML)LSTM
Loss	N/A	0.99875	**0.99711**	0.99712
Accuracy	0.520	0.531	0.550	**0.551**
F1-Score	0.52	0.542	0.551	**0.55**
Sharpe Ratio	0.478	0.493	0.563	**0.583**
Correlation	0.04	0.108	**0.1437**	0.152

Table 4. Out of Sample (Test Data) Performance Metrics

Description	(SL)MLP	(ML)MLP	(ML)RNN	(ML)LSTM
Loss	N/A	0.99775	**0.99667**	0.99688
Accuracy	0.50	**0.61**	0.6	0.59
F1-Score	0.52	0.61	**0.63**	0.58
Sharpe Ratio	0.547	0.708	**0.753**	0.746
Correlation	0.014	0.179	0.206	**0.208**

6.2 Trading Simulation Profitability

Table 5. Out of Sample (Test Data) Profitability

Description	(SL)MLP	(ML)MLP	(ML)RNN	(ML)LSTM	FTSE100
% Profit	2.9	24.1	**27.84**	23.89	10.31
Sharpe Ratio	0.547	0.708	**0.753**	0.746	0.556

MLP Figure

The results show that the RNN provides the most profitable trading strategy, as well as F1-Score. What should be taken into account is that a larger dataset with longer sequences may be better suited to a LSTM due to its ability to avoid vanishing/exploding gradients that the RNN could be subject to. What the results also show is that the Meta-Learners techniques are significantly better in predicting the direction of the returns when compared to the sub learner MLP,

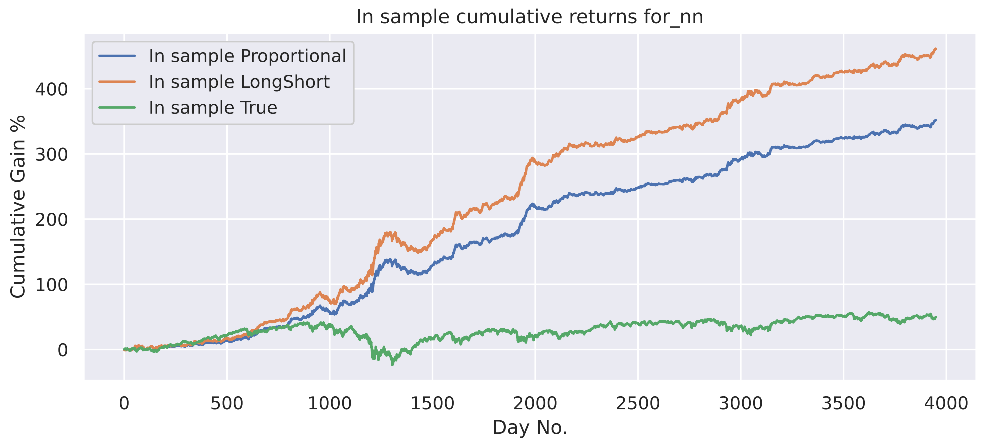

Fig. 11. Cumulative Returns on the Trading Strategies generated by the MLP vs Actual Cumulative Return of the FTSE in the Training Period

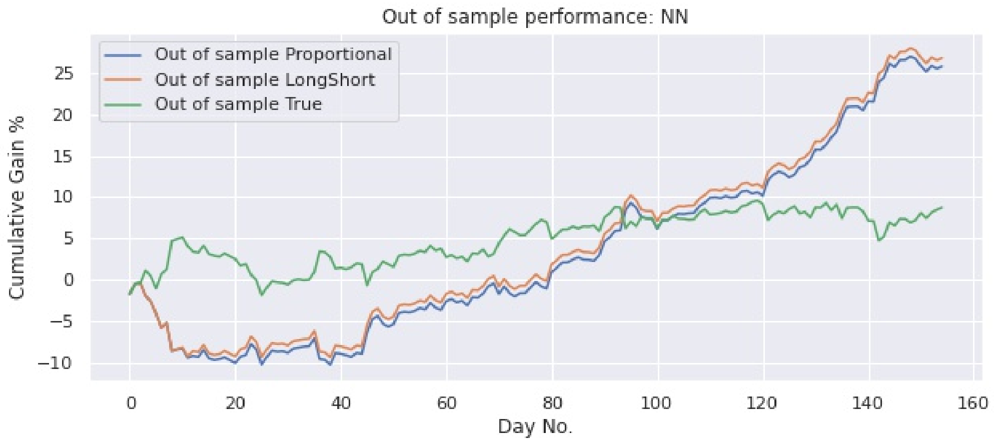

Fig. 12. Cumulative Returns on the Trading Strategies generated by the MLP vs Actual Cumulative Return of the FTSE in the Test Period

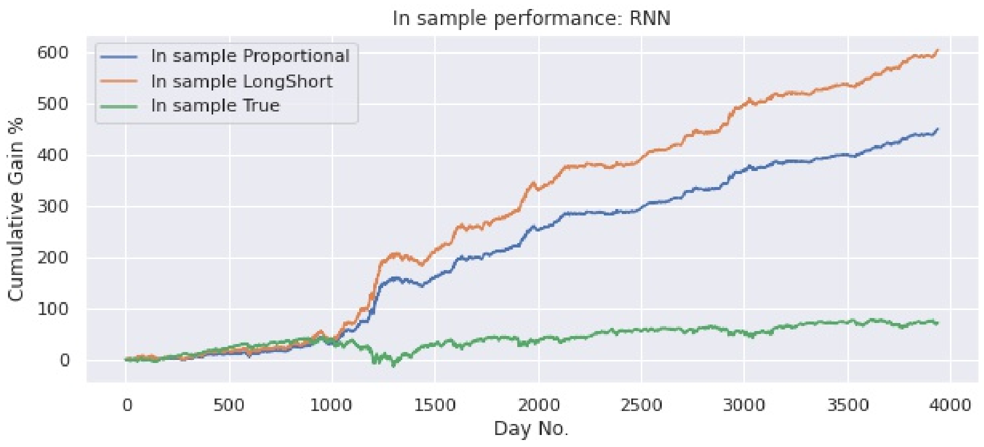

Fig. 13. Cumulative Returns on the Trading Strategies generated by the RNN vs Actual Cumulative Return of the FTSE in the Training Period

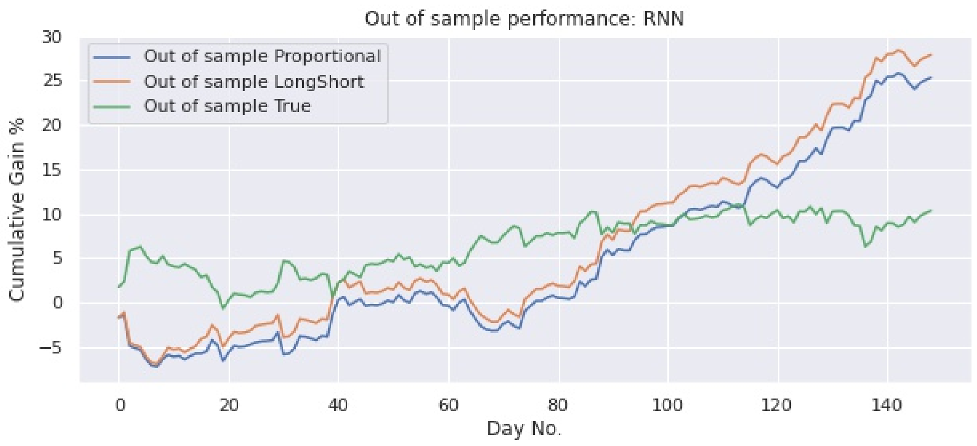

Fig. 14. Cumulative Returns on the Trading Strategies generated by the RNN vs Actual Cumulative Return of the FTSE in the Test Period

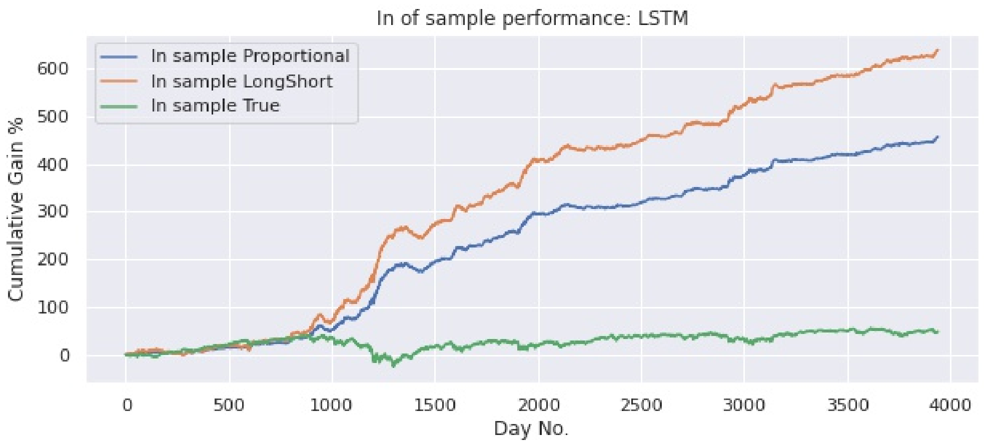

Fig. 15. Cumulative Returns on the Trading Strategies generated by the LSTM vs Actual Cumulative Return of the FTSE in the Training Period

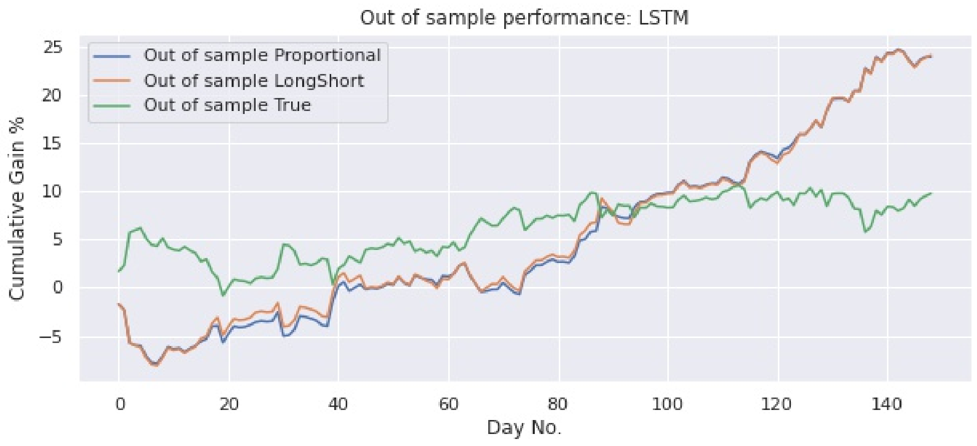

Fig. 16. Cumulative Returns on the Trading Strategies generated by the LSTM vs Actual Cumulative Return of the FTSE in the Test Period

being on average %20 better in terms of accuracy and F1-Score. The meta-learners also produce trading strategies that are almost 10 times more profitable than the one generated by the sub learner MLP, as well as having much higher Sharpe Ratios.

7 Conclusion

Experimental results show that the methodology undertaken in this study show great potential for a successful trading strategy. Further back testing is essential before any strategy can be deployed for live trading, however. What this study does show is that LSTMs and RNNs do have the ability to accurately model financial time series data, but it is essential that the right kind of data is fed into them in order to achieve results better than random guessing. PCA helps these models filter out noise and hence helps to prevent over-fitting, which leads to better generalization ability. The custom loss function used also helps the model focus more on getting the direction of the bet right than traditional loss functions like MAE and RMSE.

7.1 Challenges

Obtaining Data Obtaining sufficient data for Deep Neural networks is difficult. The data used contained 4000 observations. Typically Deep Learning Algorithms need hundreds of thousands, if not millions of rows in order to effectively learn, especially as over fitting is so easy with financial data. I experimented with data of different time periods, such as 30min data, which has approx 48 times more observations, however the models performed worse than before, likely due to the low signal to noise ratio in the 30min data. With further transformations (such as discrete wavelet transformations), it may be possible to build a better model using the 30min data.

Noisy Data Financial data is inherently noisy, meaning it is very easy to over-fit a model. The methodology used in this investigation was to truncate the log returns using a simple tanh transformation. While the models still provide encouraging results, a more sophisticated approach to removing noise and outliers should help preserve information and lead to more accurate model results.

Time Complexity Deep Neural Networks take a long time to train compared to classic ML algorithms. This meant that constraints on model building - in reducing the search space of the hyper-parameters of the models had to be introduced. As well as that, it meant that only one dataset could be investigated for this research. Given more computing resources, we could carry out this investigation on different datasets in order to get a more in depth analysis into the strengths and weaknesses of the research carried out, for example does this only work with datasets with a certain level of volatility, of a certain currency, of a certain type?(How well does it work with equities, bonds, derivatives and cryptocurrencies).

Feature Generation Creating the right features to feed into the machine learning algorithms is a science in itself. I considered including technical indicators

Data leakage It is easy while training time series data to accidentally mix test and train data up, resulting in results that appear a lot better than they are. I avoided this by using purged K-Fold cross validation while training the sub learners and by excluding the first seven records from

Data Integrity We need correct, up to date data that does not have any errors in it. It is essential that the data fed into the models is correct, up to date and reliabe. Ensuring these conditions is a challenge in itself, perhaps requiring multiple data providers to ensure consistency. data providers can get it wrong/ the data can be delayed which could eat away at any advantage/edge our ML strategy provides.

8 Future Directions

The use of Convolutional or Attention layers show good results in other areas of sequence modelling such as Natural Language Processing and can be applied to financial time series prediction. Data transformers such as ROCKET provide a promising data transformation technique that could be used to effectively model financial data. In terms of researching a profitable trading strategy, costs such as transaction fees and taxation should be taken into consideration before concluding that a trading strategy is profitable. Designing two models, whereby one specializes in predicting long and the other in predicting shorts could be another line of investigation to further this study, perhaps with a further modification of the objective function. Some models have high precision on buys and lower precision on sells, hence a model that utilizes the strengths of two different models should lead to more accurate models and stronger trading results.

Further research into feature engineering has the potential to improve these models further. The inclusion of technical analysis features, for example moving averages or momentum based strategies should be considered. The inclusion of sentiment analysis obtained through Natural Language Processing of information from news and social media websites shows great potential as a useful feature. Macroeconomic indicators such as inflation, unemployment and GDP have been found to be useful features for one step ahead prediction of major U.S. Stock indices [15] and should also be considered as potential input features for financial time series forecasting problems.

The inclusion of other sub-learners, such as tree based methods like XGBoost[4] and LightGBM[12] could further increase the predictive power of the meta learner.

Transfer Learning [16] is an area of machine learning that has not been extensively applied to Financial Time Series predicting, but is worth investigating. Transfer learning works by freezing some of the layers of an already trained Deep

Neural Network. The model is then retrained with new data and only the unfrozen layer weights are updated. Transfer learning could be especially useful for models that aim to predict intraday data with small time intervals, where there is not enough time to retrain an entire model with before a new prediction can be made.

References

1. Pelikan, M., Goldberg, D.E., 1999. BOA: The Bayesian optimization algorithm, in: In Proc. Genetic And. pp. 525–532.
2. de Prado, M.L., 2018. Advances in Financial Machine Learning. Wiley
3. Wolpert, D.H., 1992. Stacked generalization. Neural Networks 5, 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1
4. Chen, T., Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 785–794. https://doi.org/10.1145/2939672.2939785
5. Wold, S., Esbensen, K., Geladi, P., 1987. Principal component analysis. Chemometrics and Intelligent Laboratory Systems, Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists 2, 37–52. https://doi.org/10.1016/0169-7439(87)80084-9
6. Elman J.L., 1990. Finding structure in time. Cognitive science 14(2) 179-211
7. Pascanu R., Mikolov T., Bengio Y., 2013. On the difficulty of training recurrent neural networks. Proceedings of the 30th International Conference on Machine Learning 28, 1310-1318
8. Hochreiter S., Schmidhuber J. 1997. Long short-term memory. Neural Comput. 9 (8) 1735–1780.
9. Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: International Conference on Machine Learning, 1996, pp. 148–156.
10. Jiang, M., Liu, J., Zhang, L., Liu, C., 2020. An improved Stacking framework for stock index prediction by leveraging tree-based ensemble models and deep learning algorithms. Physica A: Statistical Mechanics and its Applications 541. https://doi.org/10.1016/j.physa.2019.122272
11. W.F.Sharpe. The Sharpe Ratio, The Journal of Portfolio Management Fall 1994, 21 (1) 49-58; DOI: https://doi.org/10.3905/jpm.1994.409501
12. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y., 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree, in: Advances in Neural Information Processing Systems. Curran Associates, Inc.
13. Breiman, L., 1996. Bagging predictors. Mach Learn 24, 123–140. https://doi.org/10.1007/BF00058655
14. Smyl S., 2020. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting 36 7585
15. Weng, B., Martinez, W., Tsai, Y.-T., Li, C., Lu, L., Barth, J.R., Megahed, F.M., 2018. Macroeconomic indicators alone can predict the monthly closing price of major U.S. indices: Insights from artificial intelligence, time-series analysis and hybrid models. Applied Soft Computing 71, 685–697. https://doi.org/10.1016/j.asoc.2018.07.024
16. Pan, S.J., Yang, Q., 2010. A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering 22, 1345–1359. https://doi.org/10.1109/TKDE.2009.191