Comparison of Tree-Based Ensemble Methods using Financial Time Series Data

Jiaxi Zhao - z5305338

***Abstract*—This paper benchmarks the performance of tree-based ensemble and stacking models on financial data in both a classification and regression problem. Overall, it finds that Extra Trees and Random Forest to be the best performing models and that despite superb performance on the train set, stacking models are unable to generalise well to the test set. This may be a result of the over-fitting which in turn, is due to the noisiness of financial data and the complexity of stacking models. Finding ways to effectively prevent this phenomenon is indicated as an area of future research.**

I. Introduction

Stock price prediction has fascinated financial institutions, personal investors and researchers alike. Developing any model that is able to accurately predict returns or even simply the price direction may have enormous financial benefit.

With machine learning rising in popularity, it is only natural that some of these advanced algorithms be applied to the area of stock price prediction. In particular, this benchmarking paper compares the performance of five tree-based ensemble methods: Random Forest, Extra Trees, Adaboost, Gradient Boosting and XGBoost as well as three stacking models which combine these individual ensembles with either a linear or logistic regression, neural network or support vector machine (SVM) meta-learner. Models are evaluated on both a classification and a regression problem where the goal is to either predict the accumulated five-day return (regression) or which of three classes the return falls into (classification). The features used in this study are the previous 20 days of stock returns. Data from four major indices: ASX 200, S&P 500, FTSE and Nikkei are considered. The analysis aims to provide a comprehensive evaluation of these models and detail areas for future research.

The paper is organised as follows. Section II offers a review of the literature. Section III presents the data used. Section IV details the methodology. Section V presents the results and offers some discussion and finally, section VI concludes.

II. Literature Review

Application of ensemble methods to stock market prediction date back to as early as 2011 when Tsai et al. (2011) [1] analysed the performance of majority voting and bagging methods on the Taiwanese stock market. The authors considered both homogeneous (same base learners) and heterogeneous (different base learners) models and used financial ratios and macroeconomic indicators to predict the direction of the market. The authors found that ensembles generally performed better than single models and that heterogeneous ensembles slightly outperformed homogeneous ones.

Similarly, Ballings et al. (2015) [2] benchmarked the performances of various ensemble algorithms such as the Random Forest and Adaboost against single models such as neural networks and SVMs. The goal of their study was to predict the one year price direction on a large set of European stocks. They found that the Random Forest performed best and that three of the top four performing algorithms were ensemble methods.

Some more recent studies include Weng et al. (2018) [3] who combined historical data and technical indicators with online data sources such as Google searches and related Wikipedia page visits to the prediction of stock prices of various US listed companies. The authors considered neural network and SVM ensembles, a boosted regression tree and a Random Forest. They found that the Random Forest and boosted tree outperformed the neural network and SVM ensembles.

Jiang et al. (2020) [4] considered the performance of stacking models in the prediction of three major US stock indices using a variety of macroeconomic and technical indicators. They used a variety of tree-based ensemble (Random Forest, XGBoost etc.) and neural network (LSTM, GRU etc.) base learners combined with a logistic regression meta-learner learner. They found that using lasso regularisation on the meta-learner gave the best results. However, out of the individual ensembles, they found that Extra Trees was the best performing one.

Finally Livieris et al. (2020) [5] applied ensemble methods to predicting the hourly prices of major cryptocurrencies BTC, ETH and XRP. They used deep learning models with LSTM and CNN layers as base learners, combined with either ensemble averaging, bagging or stacking. Their analysis concluded that stacking with a k-NN meta-learner resulted in the best model.

III. Data

This study uses price data from 4 major stock indices: ASX 200 (ticker: AXJO), S&P 500 (GSPC), FTSE (FTSE) and Nikkei (N225). These indices are the most widely tracked in Australia, United States, United Kingdom and Japan respectively. The data consists of daily closing prices from 01/01/2001 to 01/11/2020, a period of over 20 years or roughly 5000 trading days. The first 70% (approx. 3500 observations) was used for training and the last 30% (approx. 1500 observations) for testing. The data was obtained from Yahoo Finance [6].

It is conventional to model returns rather than the actual prices. In particular, log returns are used in this study, which are calculated via the following formula.

$$\text{return}_t(\text{in } \%) = 100 \times \log \frac{\text{price}_t}{\text{price}_{t-1}}$$

Table I below shows the summary statistics of the daily returns for each index. It can be seen that daily returns are roughly centered around 0, with FTSE having the lowest mean of -0.0046% and the GSPC having the highest mean of 0.0187%. The volatility (ie. standard deviation) varies between 1.0379% (AXJO) and 1.4864% (N225). Note that the indices used are price indices and do not take into account dividends and thus do not represent total return over the period. However, as the interest of this study is short-term prediction (five days), the data should be sufficient for this purpose.

TABLE I
SUMMARY STATISTICS: DAILY RETURNS (%)

	AXJO	GSPC	FTSE	N225
count	4985	4989	4954	4767
train count	3490	3493	3468	3337
test count	1495	1496	1486	1430
mean	0.0115	0.0187	-0.0046	0.0034
std	1.0379	1.2498	1.2019	1.4864
min	-10.203	-12.7652	-11.5117	-12.111
25%	-0.4597	-0.4549	-0.5388	-0.6941
50%	0.0497	0.0636	0.036	0.0428
75%	0.5391	0.5706	0.5802	0.7872
max	6.7665	10.9572	9.3842	13.2346

Plots of the price over time (figures 1 - 4) and density histograms of the returns (figures 5 - 8) are also shown. From the price plots, it can be seen that the training data covers a variety of market conditions. Thus, though the COVID period, which is contained in the test set, presents a volatile period of market activity, there appears to be sufficient precedence for models to learn from. In particular, the GFC in 2008 appears to represent similar conditions.

Fig. 1. AXJO price over time

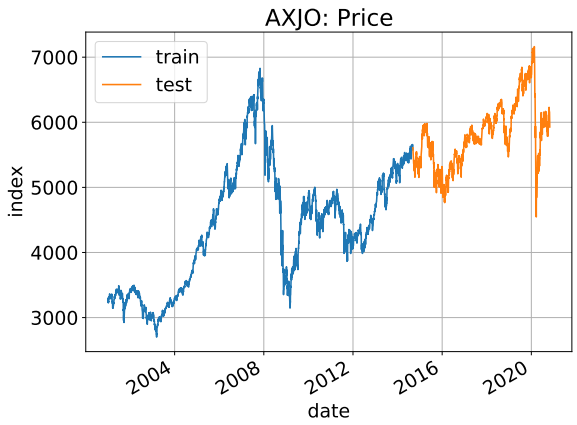


Fig. 2. GSPC price over time

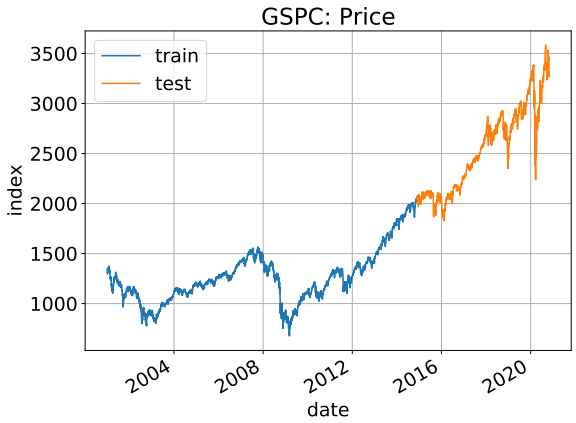


Fig. 3. FTSE price over time

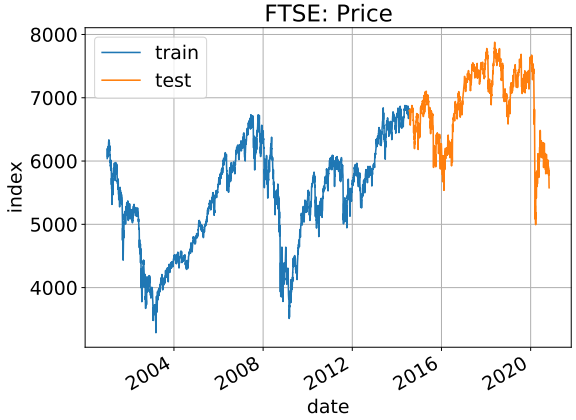
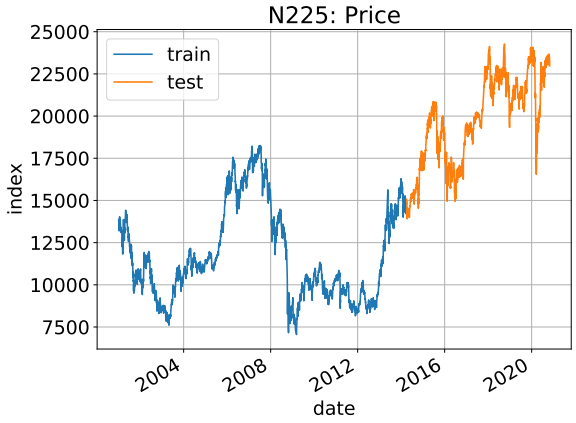


Fig. 4. N225 price over time



From the density plots, it can be seen that test returns for GSPC appears to be slightly higher on average than the train returns. Furthermore, the test returns for FTSE and N225 are slightly less volatile. However, overall, distributions still appear to be quite similar for each index.

Fig. 5. AXJO return distribution

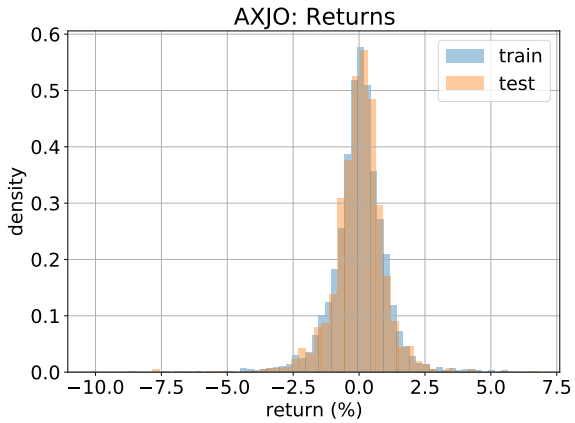


Fig. 6. GSPC return distribution

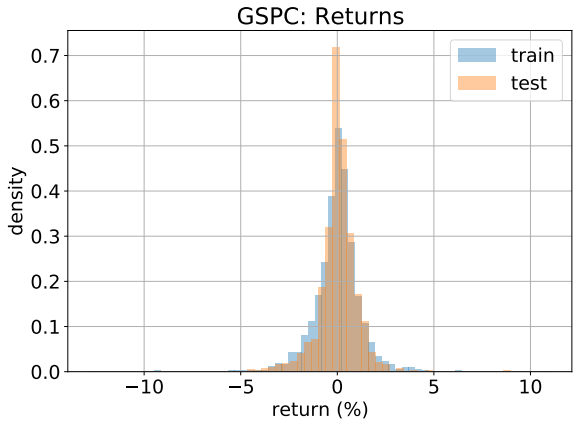


Fig. 7. FTSE return distribution

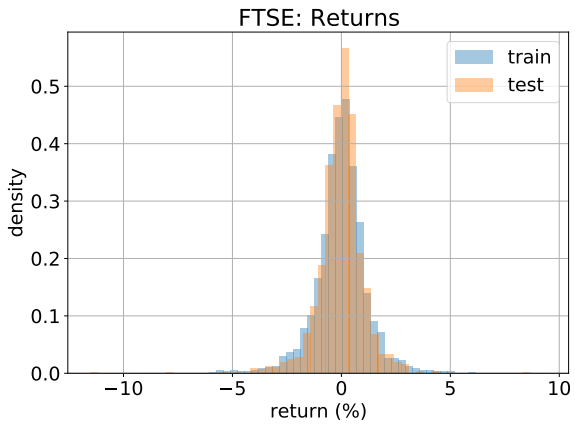
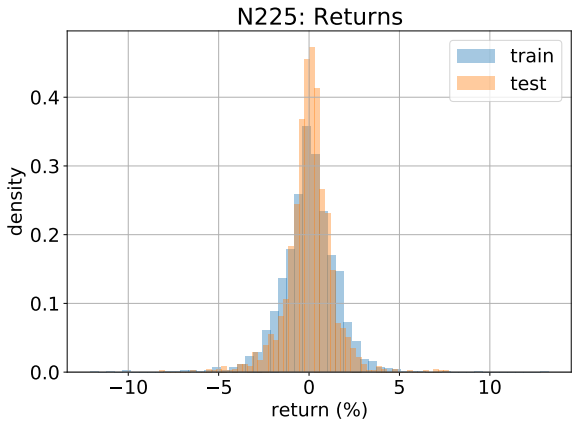


Fig. 8. N225 return distribution



A. *Classification Data*

The first problem this paper considers is a classification problem. In particular, for each of the four indices separately, it considers predicting whether the five day performance (target) is 'down', 'unchanged' or 'up' using the previous 20 daily returns (features). Performance is considered 'down' if the five day return (accumulated) is below the 33rd percentile of such returns, 'unchanged' if it falls between the 33rd and 67th percentiles and 'up' if it falls above the 67th. These class cut-offs were obtained using the train data only. Tables II and III show the cut-offs used and the distribution of the three classes. Evidently, the train data is completely balanced given that it was used to compute the cut-offs. There is a slight imbalance in the test set as we see a higher proportion of observations in the 'unchanged' and 'up' categories.

TABLE II
CLASSIFICATION: CLASS CUTOFFS

	Down Cut-off	Up Cut-off
AXJO	-0.5185	0.9391
GSPC	-0.5591	1.0227
FTSE	-0.6932	0.9309
N225	-1.1344	1.3532

TABLE III
CLASSIFICATION: CLASS DISTRIBUTIONS

	Down	Unchanged	Up
AXJO: train	1155	1155	1155
AXJO: test	489	535	448
GSPC: train	1156	1156	1156
GSPC: test	386	613	474
FTSE: train	1148	1147	1148
FTSE: test	450	571	442
N225: train	1104	1104	1104
N225: test	345	629	433

B. *Regression Data*

The paper also considers a regression problem which is to predict the actual five day return rather than which category

it falls into, Again, this is done using the previous 20 daily returns as features.

Summary statistics of the target are shown in Tables IV and V below. It can be seen that the test data does vary from the train data. In particular, GSPC and N225 mean returns are substantially higher in the test set and N225 returns appear to be less volatile.

TABLE IV
REGRESSION: 5-DAY RETURNS (%), TRAIN

	AXJO	GSPC	FTSE	N225
count	3465	3468	3443	3312
mean	0.0658	0.0589	-0.0103	-0.0182
std	2.1887	2.5915	2.5514	3.3086
min	-17.0163	-20.2607	-23.6316	-27.8844
25%	-1.0039	-1.1395	-1.2395	-1.8945
50%	0.2545	0.2566	0.1633	0.2262
75%	1.3243	1.4666	1.3542	1.9953
max	12.0161	17.4887	16.6884	24.0963

TABLE V
REGRESSION: 5-DAY RETURNS (%), TEST

	AXJO	GSPC	FTSE	N225
count	1472	1473	1463	1407
mean	0.0337	0.1604	-0.0418	0.1065
std	2.2414	2.3569	2.4033	2.8924
min	-18.7052	-19.8044	-24.7069	-17.4281
25%	-0.9305	-0.649	-1.0591	-1.0887
50%	0.2036	0.3868	0.1369	0.303
75%	1.2521	1.3273	1.134	1.6826
max	13.0828	16.0395	12.1254	15.8171

IV. METHODOLOGY

In this section, a detailed description of the methodology is given. As previously mentioned, this paper considers both a classification and a regression problem. To reiterate, the classification problem is to predict, for each index separately, if the five day index performance is 'up', 'down' or 'unchanged' whilst the regression problem is to predict the accumulated five day return. Both use the past 20 daily returns as features.

The analysis considers the prediction performance of five tree-based ensemble methods, namely: Random Forest (RF), Extra Trees (ET), Adaboost (ADA), Gradient Boosting (GB) and XGBoost (XGB). Additionally, it considers the performance of three stacking models, each of which use all five ensemble models as base learners and combines them with either a linear or logistic regression (Simple Stacker), neural network (Network Stacker) or SVM (SVM Stacker) meta-learner. These above ensemble and stacking models were also compared against simple logistic and linear regression models (Simple) for reference.

The stages of the analysis were organised as follows:

- Hyper-parameter tuning for ensemble methods
- Hyper-parameter tuning for meta-learners
- Performance evaluation of the models

Models were implemented in Python using the following key packages: Scikit-Learn [7], Tensorflow [8] and XGBoost [9]. Additionally, Geron's (2019) [10] Hands-On Machine Learning with Scikit-Learn and Tensorflow was consulted to develop the code.

A. Hyper-parameter tuning for ensemble methods

For each ensemble method, the following hyper-parameterisations were considered:

- **RF**:
  - max depth: 2, 4, 8
  - max features: 2, 4, 8
- **ET**:
  - max depth: 2, 4, 8
  - max features: 2, 4, 8
- **ADA**:
  - learning rate: 0.01, 0.1, 1.0, 5.0
- **GB**:
  - learning rate: 0.01, 0.1, 1.0
  - max depth: 2, 4, 8
  - max features: 2, 4, 8
- **XGB**:
  - learning rate: 0.01, 0.1, 1.0
  - max depth: 2, 4, 8

Hyper-parameters not detailed above were left as their default value from their respective packages. Additionally, each ensemble model was fit with 400 estimators.

To find the optimal hyper-parameterisation, a grid search using 10-fold cross validation was performed using the train data only. The best hyper-parameters were identified for each index (AXJO, GSPC, FTSE, N225) and each problem type (classification or regression) separately.

After the best hyper-parameters were identified, model predictions on the train data set were obtained. These were fed into the meta-learner for their hyper-parameter tuning.

B. Hyper-parameter tuning for meta-learners

For meta-learner tuning, predictions from the base learners were frozen and essentially considered as input features. In the classification problem, ensemble predictions consisted of predicted probabilities for each of the three classes. Thus, with five ensemble methods, a total of 15 features were used by each meta-learner. For the regression problem, the output was simply the predicted return so there were only five features.

1) Simple: Linear and logistic regression models did not have any hyper-parameters that required tuning.

2) Network: The 'base' neural network meta-learner consisted of either 15 (classification) or five (regression) input neurons, a single hidden layer with 10 hidden neurons and either three (classification) or one (regression) output neuron(s). Furthermore, after each hidden layer, a dropout layer with a dropout ratio of 0.2 was used. A ReLU activation function was used for each hidden layer and either a softmax (classification) or linear (regression) activation function was

used for the output layer. The loss used was either log-loss for classification or mean squared error for regression. Adam, with default parameter values, was used for optimisation.

An early stopping callback was also implemented for the training procedure. First, 25% of the training data was randomly split off for validation and the remainder kept for training. After each epoch, model performance was evaluated on the validation data. If the validation loss failed to improve for 25 epochs, training would cease and the weights rolled back to the best model.

The hyper-parameters tuned were:

- number of hidden neurons: 5, 10 or 20
- number of hidden layers: 1, 2, or 3
- dropout ratio: 0.0, 0.1, 0.2, or 0.3

Tuning was conducted using a waterfall procedure where first the number of hidden neurons (in a single hidden layer) was identified, followed by the number of hidden layers and finally the dropout ratio. After each round of investigation, the best hyper-parameterisation was carried into the next. Ten-fold cross validation using the train data only was again used to evaluate the performance of each hyper-parameter set.

3) SVM: The only hyper-parameter tuned for the SVM meta-learner was the regularisation parameter. The values tested were 0.01, 0.1, 1, 10 and 100. Other parameters were left as their default values. Similar to the ensemble methods, evaluation was done using a grid search with ten-fold cross-validation.

C. Evaluation of models

After the best hyper-parameters were identified, each model was fit 10 times and evaluated on both the train and test sets. For stacking models each base learner was also retrained each iteration before their predictions were fed to the meta-learner to be used in their training.

For classification, the evaluation metric used was accuracy whereas root mean squared error (RMSE) was used for regression. For each index, problem type and train and test sets separately, the mean, standard deviation and 95% confidence interval of the evaluation metric was computed. Furthermore, models were ranked one through nine (ie. five ensemble methods and three stacking models and the simple model) with one representing the best performing model.

V. Results and Discussion

Overall, the results across each index were fairly similar so only the results for AXJO are discussed at length in this section. References to the other indices are made when comparing the overall performance of each model.

A. Classification

Table VI shows the obtained hyper-parameters for AXJO classification models.

TABLE VI
Classification Hyper-parameters for AXJO

Model	Parameters
RF	max_depth: 8
	max_features: 8
ET	max_depth: 8
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.01
	max_depth: 4
	max_features: 2
XGB	learning_rate: 0.01
	max_depth: 2
Network	neurons: 20
	layers: 3
	dropout: 0.1
SVM	regularisation: 100

Classification performance on the train and test sets separately for AXJO are reported below in Tables VII and VIII. Box plots (figures 9 and 10) have also been included for easier visualisation.

TABLE VII
Train Accuracy: AXJO

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.7841	0.0027	0.7822	0.786	4
ET	0.6399	0.0035	0.6373	0.6424	6
ADA	0.4355	0	0.4355	0.4355	8
GB	0.6554	0.0027	0.6535	0.6573	5
XGB	0.4938	0	0.4938	0.4938	7
Simple Stacker	0.8473	0.004	0.8445	0.8502	3
Network Stacker	0.9056	0.0034	0.9032	0.908	1
SVM Stacker	0.905	0.0046	0.9017	0.9083	2
Simple	0.3974	0	0.3974	0.3974	9

TABLE VIII
Test Accuracy: AXJO

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.408	0.0053	0.4042	0.4118	4
ET	0.4283	0.0041	0.4254	0.4313	1
ADA	0.3927	0	0.3927	0.3927	5
GB	0.4119	0.0035	0.4094	0.4144	2
XGB	0.411	0	0.411	0.411	3
Simple Stacker	0.3825	0.0075	0.3771	0.3879	6
Network Stacker	0.3567	0.0117	0.3483	0.3651	8
SVM Stacker	0.3484	0.0078	0.3428	0.3539	9
Simple	0.3607	0	0.3607	0.3607	7

Fig. 9. AXJO classification train results

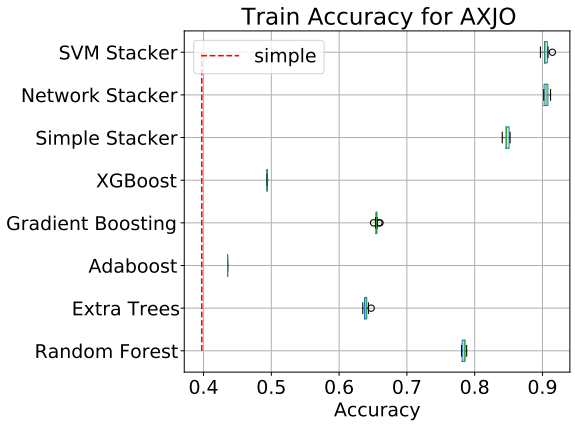
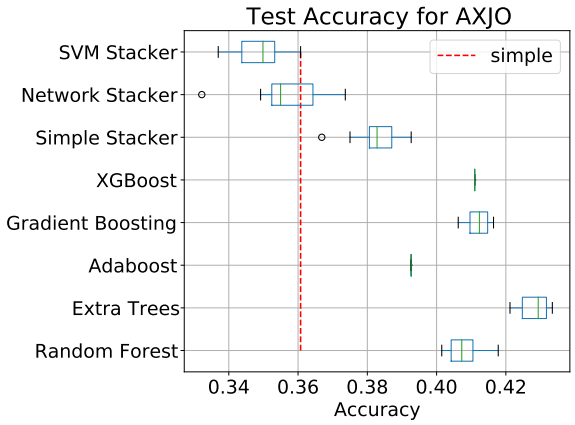


Fig. 10. AXJO classification test results



The mean train accuracy scores for AXJO range from 0.3974 (Simple) to 0.9056 (Network Stacker). For context, since the train data is completely balanced, a model predicting randomly would have an accuracy of approximately 0.33. Standard deviations are all fairly small (less than 0.005) so it is sufficient to only compare the means. Unsurprisingly, stacking algorithms performed the best, with the Network Stacker, SVM Stacker (0.905) and Simple Stacker (0.8473) holding the top three ranks. Of the ensembles, RF (0.7841) performed the best, followed by GB (0.6554) whilst ADA performed the worst (0.4355). All stacking and ensemble models performed better than Simple.

However, a completely different story is seen in the test accuracy scores. As expected, scores were much lower, ranging from 0.3484 (SVM Stacker) to 0.4283 (ET). Stacking models performed extremely poorly with the SVM Stacker (0.3484) and Network Stacker (0.3567) having the two lowest ranks. Additionally, they even performed worse than Simple (0.3607) and were little better than guessing at random. The Simple Stacker (0.3835) did not do much better. Of the ensemble methods, ET (0.4283) performed the best, followed by GB (0.4119) whilst ADA (0.3927) performed the worst. It is interesting to note that all ensemble methods outperformed Simple but the majority of stacking models did not.

The following tables detail the classification ranks for each index as well as the average rank.

TABLE IX
CLASSIFICATION RANKS: TRAIN

	AXJO	GSPC	FTSE	N225	average
RF	4	6	4	4	4.5
ET	6	7	6	6	6.25
ADA	8	8	8	8	8
GB	5	5	5	5	5
XGB	7	4	7	7	6.25
Simple Stacker	3	3	3	3	3
Network Stacker	1	2	1	1	1.25
SVM Stacker	2	1	2	2	1.75
Simple	9	9	9	9	9

TABLE X
CLASSIFICATION RANKS: TEST

	AXJO	GSPC	FTSE	N225	average
RF	4	4	2	2	3
ET	1	2	1	1	1.25
ADA	5	3	3	4	3.75
GB	2	1	5	3	2.75
XGB	3	5	4	6	4.5
Simple Stacker	6	6	7	5	6
Network Stacker	8	8	8	9	8.25
SVM Stacker	9	9	9	8	8.75
Simple	7	7	6	7	6.75

As seen above, results are fairly consistent across the indices. In particular, stacking models performed well on the train set but failed to generalise and were often outperformed by the Simple model on the test set. ET had the highest average test ranking, placing first on AXJO, FTSE and N225 and second on GSPC. This is despite its average performance on the train set. GB ranked second overall and XGB showed the worst performance out of the ensemble methods.

Results are somewhat surprising given that it is expected that stacking models perform better than their individual base learners. If only the train set were examined, this would be true. However, the test set tells the opposite story and their inability to generalise well is undoubtedly due to over-fitting. In particular, financial returns are extremely noisy and so it is possible that a single weak learner is able to capture all of the existing patterns. By then feeding their predictions to a meta-learner all that is left is for the meta-learner capture noise.

An argument could be made for better hyper-parameter tuning. For example, the AXJO Network Stacker uses a meta-learner with three hidden layers of 20 hidden neurons which appears to be too complex a model. However, for other indices, simpler networks are used (see appendix) and an effort has been made to regularise the model using dropout and early stopping. Furthermore, even the Simple Network, which only uses a logistic regression meta-learner, fails to generalise well.

Thus, it appears that better hyper-parameter tuning is unable to completely solve the problem of over-fitting and may just be result of the data used.

Finally, the noisiness of the data may also explain why ET was the best performing model. Relative to other ensemble methods, the ET method is weak given that it is a tree-based method that determines the split points for each feature randomly. This acts as a form of regularisation and prevents the tree from over-fitting. In usual instances where patterns in the data are more clear, this simplicity prevents the tree from learning effectively. However, in the case of financial data, preventing over-fitting appears to be the more important consideration.

B. Regression

Table XI below shows the obtained hyper-parameters for AXJO regression models.

TABLE XI
REGRESSION HYPER-PARAMETERS FOR AXJO

Model	Parameters
RF	max_depth: 2 max_features: 2
ET	max_depth: 2 max_features: 2
ADA	learning_rate: 0.01
GB	learning_rate: 0.01 max_depth: 2 max_features: 2
XGB	learning_rate: 0.1 max_depth: 2
Network	neurons: 10 layers: 1 dropout: 0.0
SVM	regularisation: 1

Regression performance on the train and test sets separately for AXJO are reported below in Tables XII and XIII below. Box plots (figures 11 and 12) have also been included for easier visualisation.

Fig. 11. AXJO regression train results

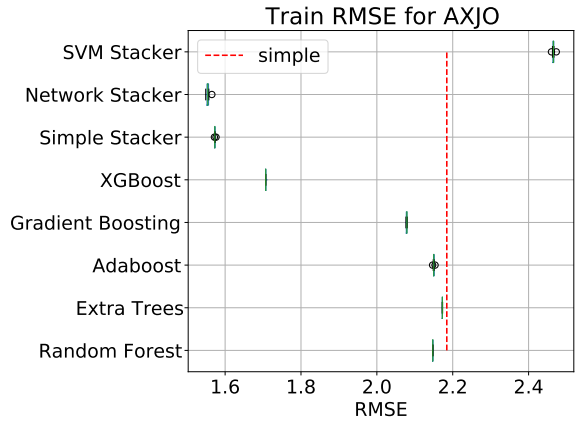
TABLE XII
TRAIN RMSE: AXJO

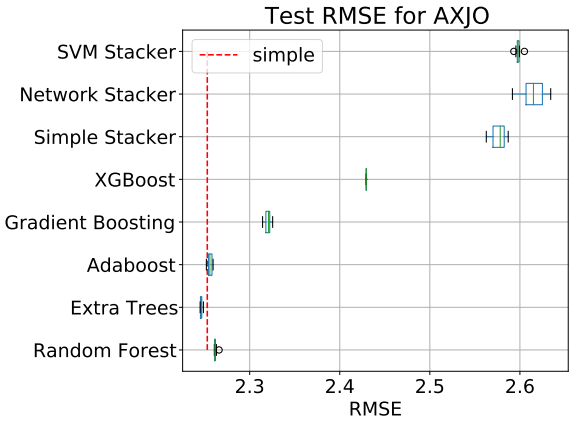
	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.1475	0.0003	2.1473	2.1477	5
ET	2.1726	0.0004	2.1723	2.173	7
ADA	2.1501	0.0017	2.1489	2.1513	6
GB	2.0786	0.0013	2.0777	2.0795	4
XGB	1.7071	0	1.7071	1.7071	3
Simple Stacker	1.5731	0.0012	1.5723	1.5739	2
Network Stacker	1.5545	0.0046	1.5512	1.5579	1
SVM Stacker	2.4655	0.0033	2.4632	2.4678	9
Simple	2.1844	0	2.1844	2.1844	8

Fig. 12. AXJO regression test results

TABLE XIII
TEST RMSE: AXJO

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.2619	0.0016	2.2608	2.2631	4
ET	2.2464	0.0012	2.2456	2.2473	1
ADA	2.256	0.0025	2.2542	2.2577	3
GB	2.3203	0.0034	2.3178	2.3227	5
XGB	2.4294	0	2.4294	2.4294	6
Simple Stacker	2.5761	0.0083	2.5702	2.5821	7
Network Stacker	2.6152	0.0128	2.6061	2.6244	9
SVM Stacker	2.5981	0.0031	2.5959	2.6003	8
Simple	2.2529	0	2.2529	2.2529	2





The mean train RMSE scores for AXJO range between 2.4655 (SVM Stacker) to 1.5545 (Network Stacker). For context, the standard deviation of five-day returns for AXJO on the train set is 2.1887. Once again, since all standard deviations are fairly small (less than 0.005), it is sufficient to compare the means. The Network Stacker and Simple stacker (1.5731) performed the best out of all models whilst the SVM Stacker performed the worst. However, the fact that the SVM Stacker performed substantially worse than Simple (2.1844) indicates it may have been misparameterised. In terms of ensemble methods, XGB (1.7071) performed the best, followed by GB (2.0786) whilst ET (2.1726) performed the

worst. All stacking and ensemble methods. with the exception of the SVM Stacker, performed better than Simple.

Once again, there is a significant drop-off in performance in the test set. Mean test RMSE scores range between 2.6152 (Network Stacker) and 2.2464 (ET). Furthermore, Simple (2.2529) outperformed all other models with with the exception of ET, which itself was only marginally better. Stacking models in particular, saw the largest drop-off in performance with Network Stacker, SVM Stacker (2.5981) and Simple Stacker (2.5761) being the three worst performing. Apart from ET, ADA (2.2456) performed the best out of the ensemble methods and XGB (2.4294) performed the worst.

The following tables detail the regression ranks for each index as well as the average rank.

TABLE XIV
REGRESSION RANKS: TRAIN

	AXJO	GSPC	FTSE	N225	average
RF	5	6	6	7	6
ET	7	7	4	8	6.5
ADA	6	5	7	6	6
GB	4	4	5	5	4.5
XGB	3	3	3	3	3
Simple Stacker	2	2	1	1	1.5
Network Stacker	1	1	2	2	1.5
SVM Stacker	9	9	9	4	7.75
Simple	8	8	8	9	8.25

TABLE XV
REGRESSION RANKS: TEST

	AXJO	GSPC	FTSE	N225	average
RF	4	2	1	2	2.25
ET	1	3	3	1	2
ADA	3	4	4	4	3.75
GB	5	5	2	5	4.25
XGB	6	6	6	6	6
Simple Stacker	7	8	9	9	8.25
Network Stacker	9	9	8	8	8.5
SVM Stacker	8	7	7	7	7.25
Simple	2	1	5	3	2.75

As seen above, train performances are fairly consistent across the indices. Network Stacker and Simple Stacker performed the best, ranking either one or two for each index whilst SVM Stacker performed the worst for most indices. Of the ensemble methods, XGB performed the best, followed by GB whilst ET performed the worst. This is alsmost exactly in line with the results from AXJO. All models performed better than Simple on average, though the results from the SVM Stacker appear to be skewed by its performance on N225.

For the test set, results across the indices are more diverse. Overall, ET performed the best, followed by RF and then Simple. That is, a simple linear regression model performed better than all but two of the ensemble and stacking models. The three worst performing models were the Network Stacker, Simple Stacker and SVM Stacker, in line with results from AXJO.

It is surprising that stacking models performed substantially worse on the test set but the results are consistent with what was seen in the classification problem. Ignoring the SVM Stacker which may have been misparametrised, the sharp drop-off from the train performance to test performance for the Network and Simple Stackers is clear evidence of over-fitting. The reason for this may be similar to the classification case.

For ensemble methods, the over-fitting problem appears to be much more of an issue in the regression problem than the classification one. In the classification problem, all were able to outperform the Simple model on the test set, despite the drop-off in performance. In the regression case, only RF and ET outperformed the Simple model, indicating that ADA, GB and XGB also suffered substantially from over-fitting. This should not be entirely unexpected, given that predicting the actual return value is a much more difficult task than classifying them into one of three categories, as the target variable is much noisier. Furthermore, the top two performing algorithms for the regression problem ET and RF, ranked one and three respectively on the classification problem. Both are bagging algorithms whilst the other ensemble methods use boosting. Thus suggests a general trend that bagging algorithms are more equipped for dealing with financial data.

VI. CONCLUSION

Overall, the analysis found that ET and RF outperformed other ensembles ensembles and stacking models. Though these two algorithms generally performed sub-par on the train set, their ability to generalise better to the test set made them the favourite. The superior performance of these two models follow the findings of Ballings et al. (2015) [2] and Weng et al. (2018) [3], who also found RF to be the best and Jiang et al. (2020) [4], who found that ET to be the best performing individual ensemble.

However, possibly due to the noisy nature of financial data, the analysis did not support the use of stacking models. In particular, a general trend of superb performance on the train set and then extremely poor performance on the test set indicated over-fitting. This was despite attempts at regularising the model using dropouts and early stopping for the Network Stacker and tuning the regularisation parameter for the SVM Stacker. This appears to contradict the findings of Jiang et al. (2020) [4] and Livieris et al. [5] who both reported a stacking model as their best performing one. Differences may be due to different base and meta learners, the use of technical and macroeconomic features and the use of target smoothing to remove noise.

Undoubtedly, stacking models are extremely powerful as demonstrated by their performance on the train set. The challenge appears to be to prevent them from over-fitting and future research dedicated to investigating this may be valuable. In particular, this analysis used 400 estimators for each ensemble in the case of evaluating the ensemble individually and in the case of the stacking model. This may not be the best approach but rather the number of estimators should decrease when they are used in a stacking model.

Furthermore, future analysis verifying the robustness of these findings using a wider variety of evaluation metrics (such as log-loss for classification) and incorporating a more diverse range of features (ie. macroeconomic and technical indicators) may also be of value.

VII. Appendix

A. Hyper-paramaters for GSPC, FTSE and N225

TABLE XVI
CLASSIFICATION HYPER-PARAMETERS FOR GSPC

Model	Parameters
RF	max_depth: 2
	max_features: 8
ET	max_depth: 2
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.01
	max_depth: 2
	max_features: 8
XGB	learning_rate: 0.01
	max_depth: 8
Network	neurons: 20
	layers: 1
	dropout: 0.2
SVM	regularisation: 100

TABLE XVII
CLASSIFICATION HYPER-PARAMETERS FOR FTSE

Model	Parameters
RF	max_depth: 8
	max_features: 8
ET	max_depth: 8
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.1
	max_depth: 2
	max_features: 4
XGB	learning_rate: 0.01
	max_depth: 2
Network	neurons: 20
	layers: 2
	dropout: 0.0
SVM	regularisation: 100

TABLE XVIII
CLASSIFICATION HYPER-PARAMETERS FOR N225

Model	Parameters
RF	max_depth: 8
	max_features: 2
ET	max_depth: 8
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.01
	max_depth: 4
	max_features: 2
XGB	learning_rate: 0.1
	max_depth: 2
Network	neurons: 20
	layers: 2
	dropout: 0.1
SVM	regularisation: 100

TABLE XIX
REGRESSION HYPER-PARAMETERS FOR GSPC

Model	Parameters
RF	max_depth: 2
	max_features: 2
ET	max_depth: 2
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.01
	max_depth: 2
	max_features: 2
XGB	learning_rate: 0.1
	max_depth: 2
Network	neurons: 20
	layers: 2
	dropout: 0.1
SVM	regularisation: 0.01

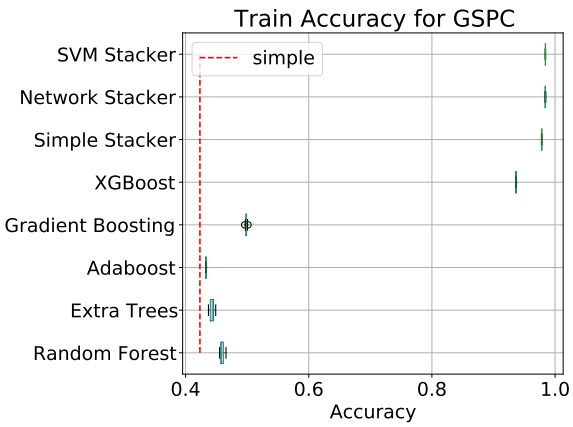
TABLE XX
REGRESSION HYPER-PARAMETERS FOR FTSE

Model	Parameters
RF	max_depth: 4
	max_features: 8
ET	max_depth: 8
	max_features: 8
ADA	learning_rate: 0.01
GB	learning_rate: 0.01
	max_depth: 2
	max_features: 8
XGB	learning_rate: 0.1
	max_depth: 2
Network	neurons: 10
	layers: 1
	dropout: 0.3
SVM	regularisation: 0.01

TABLE XXI
REGRESSION HYPER-PARAMETERS FOR N225

Model	Parameters
RF	max_depth: 2 max_features: 2
ET	max_depth: 2 max_features: 4
ADA	learning_rate: 0.01
GB	learning_rate: 0.01 max_depth: 2 max_features: 8
XGB	learning_rate: 0.1 max_depth: 2
Network	neurons: 5 layers: 1 dropout: 0.1
SVM	regularisation: 10

Fig. 13. GSPC classification train results



B. *Classification results for GSPC, FTSE and N225*

Fig. 14. GSPC classification test results

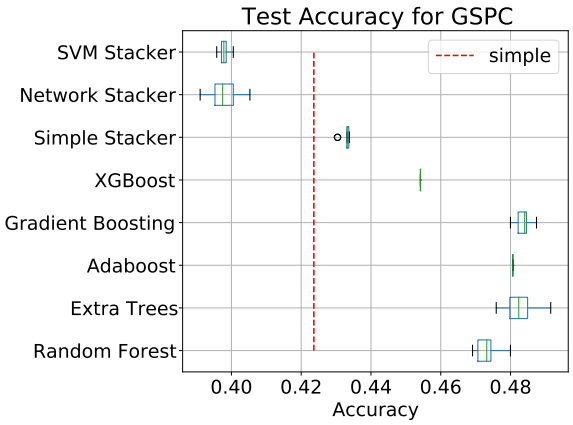


TABLE XXII
TRAIN ACCURACY: GSPC

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.4595	0.0034	0.4571	0.4619	6
ET	0.443	0.0038	0.4403	0.4458	7
ADA	0.4331	0	0.4331	0.4331	8
GB	0.4987	0.0016	0.4975	0.4998	5
XGB	0.9366	0	0.9366	0.9366	4
Simple Stacker	0.9782	0.0003	0.978	0.9784	3
Network Stacker	0.9841	0.0005	0.9838	0.9845	2
SVM Stacker	0.9843	0.0002	0.9841	0.9845	1
Simple	0.4233	0	0.4233	0.4233	9

TABLE XXIII
TEST ACCURACY: GSPC

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.4734	0.0036	0.4708	0.4759	4
ET	0.4827	0.0045	0.4795	0.4859	2
ADA	0.4807	0	0.4807	0.4807	3
GB	0.4836	0.0022	0.482	0.4852	1
XGB	0.4542	0	0.4542	0.4542	5
Simple Stacker	0.4328	0.0013	0.4319	0.4337	6
Network Stacker	0.398	0.0047	0.3946	0.4013	8
SVM Stacker	0.3979	0.0014	0.3969	0.3989	9
Simple	0.4236	0	0.4236	0.4236	7

TABLE XXIV
TRAIN ACCURACY: FTSE

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.7816	0.0041	0.7787	0.7845	4
ET	0.639	0.0023	0.6374	0.6406	6
ADA	0.4284	0	0.4284	0.4284	8
GB	0.6799	0.0019	0.6786	0.6813	5
XGB	0.4984	0	0.4984	0.4984	7
Simple Stacker	0.8578	0.0035	0.8553	0.8602	3
Network Stacker	0.9243	0.0031	0.922	0.9265	1
SVM Stacker	0.9156	0.0036	0.913	0.9181	2
Simple	0.4089	0	0.4089	0.4089	9

TABLE XXV
TEST ACCURACY: FTSE

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.3988	0.0036	0.3962	0.4013	2
ET	0.4154	0.0033	0.413	0.4178	1
ADA	0.3964	0	0.3964	0.3964	3
GB	0.3716	0.006	0.3673	0.3758	5
XGB	0.3814	0	0.3814	0.3814	4
Simple Stacker	0.3645	0.008	0.3588	0.3702	7
Network Stacker	0.353	0.0075	0.3477	0.3584	8
SVM Stacker	0.3491	0.0097	0.3422	0.3561	9
Simple	0.365	0	0.365	0.365	6

TABLE XXVI
TRAIN ACCURACY: N225

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.8241	0.0042	0.8211	0.8271	4
ET	0.6935	0.0033	0.6911	0.6959	6
ADA	0.4155	0	0.4155	0.4155	8
GB	0.6964	0.0026	0.6945	0.6983	5
XGB	0.6821	0	0.6821	0.6821	7
Simple Stacker	0.8269	0.0043	0.8238	0.8299	3
Network Stacker	0.9251	0.0066	0.9204	0.9298	1
SVM Stacker	0.9103	0.0048	0.9069	0.9137	2
Simple	0.3835	0	0.3835	0.3835	9

Fig. 15. FTSE classification train results

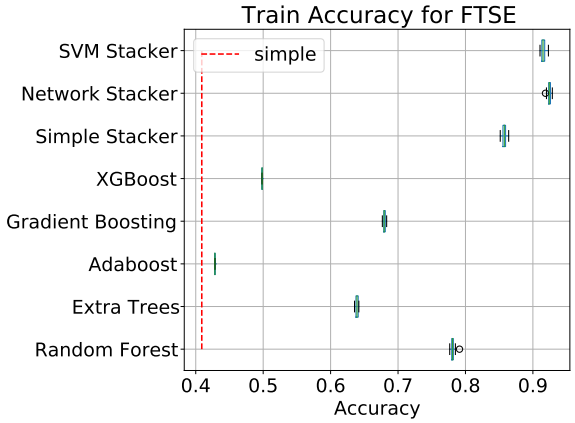


TABLE XXVII
TEST ACCURACY: N225

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	0.4539	0.0043	0.4508	0.4571	2
ET	0.4551	0.0045	0.4519	0.4583	1
ADA	0.4435	0	0.4435	0.4435	4
GB	0.4495	0.0029	0.4474	0.4516	3
XGB	0.4343	0	0.4343	0.4343	6
Simple Stacker	0.4363	0.004	0.4335	0.4392	5
Network Stacker	0.407	0.0107	0.3993	0.4147	9
SVM Stacker	0.4071	0.009	0.4007	0.4135	8
Simple	0.4094	0	0.4094	0.4094	7

Fig. 16. FTSE classification test results

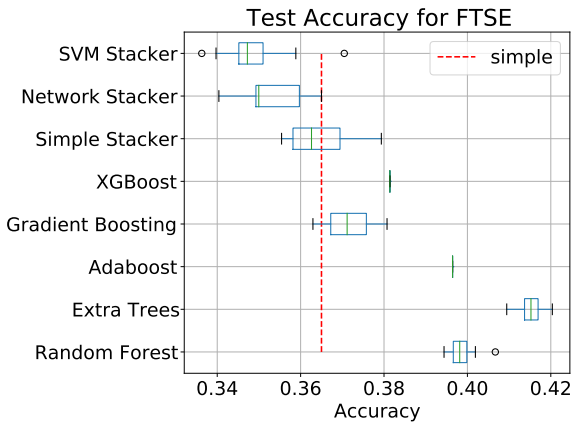


Fig. 17. N225 classification train results

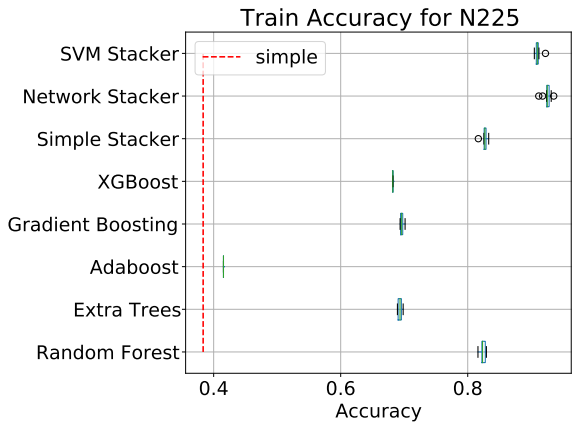


Fig. 18. N225 classification test results

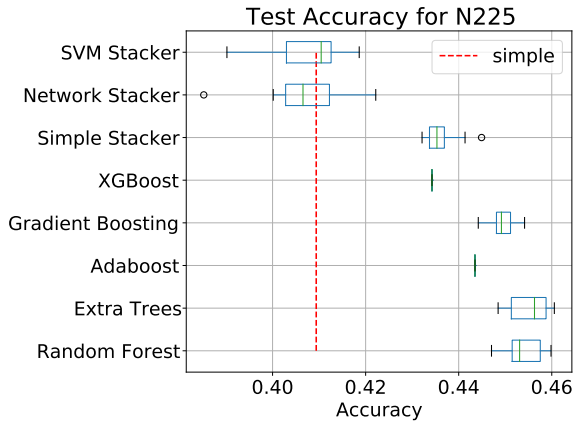
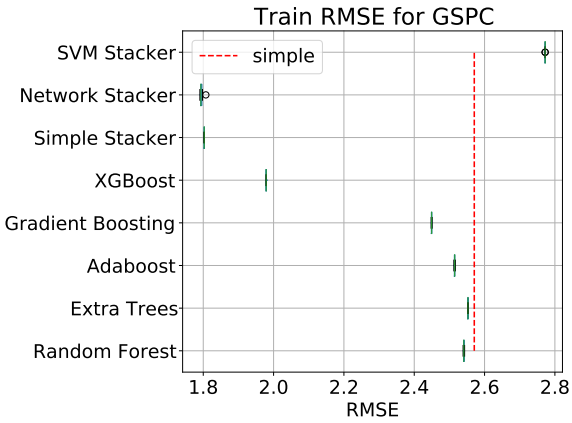


Fig. 19. GSPC regression train results



C. Regression results for GSPC, FTSE and N225

Fig. 20. GSPC regression test results

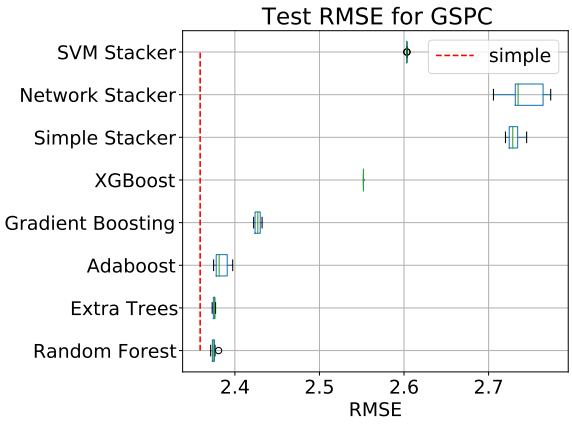


TABLE XXVIII
TRAIN RMSE: GSPC

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.5412	0.0012	2.5404	2.5421	6
ET	2.5527	0.0008	2.5522	2.5533	7
ADA	2.5148	0.0013	2.5139	2.5158	5
GB	2.4495	0.001	2.4487	2.4502	4
XGB	1.9789	0	1.9789	1.9789	3
Simple Stacker	1.8027	0.0007	1.8022	1.8032	2
Network Stacker	1.7952	0.0048	1.7917	1.7986	1
SVM Stacker	2.7723	0.0001	2.7722	2.7723	9
Simple	2.5707	0	2.5707	2.5707	8

TABLE XXIX
TEST RMSE: GSPC

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.375	0.0026	2.3732	2.3769	2
ET	2.3754	0.0014	2.3744	2.3764	3
ADA	2.3843	0.0084	2.3783	2.3904	4
GB	2.4269	0.0036	2.4244	2.4295	5
XGB	2.5521	0	2.5521	2.5521	6
Simple Stacker	2.7298	0.0075	2.7244	2.7352	8
Network Stacker	2.7423	0.023	2.7259	2.7588	9
SVM Stacker	2.6034	0.0001	2.6033	2.6034	7
Simple	2.3589	0	2.3589	2.3589	1

TABLE XXX
TRAIN RMSE: FTSE

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.4254	0.0009	2.4248	2.4261	6
ET	2.3513	0.0041	2.3484	2.3542	4
ADA	2.48	0.0014	2.479	2.481	7
GB	2.3911	0.0012	2.3903	2.392	5
XGB	2.0262	0	2.0262	2.0262	3
Simple Stacker	1.8676	0.0006	1.8672	1.868	1
Network Stacker	1.8796	0.0136	1.8699	1.8893	2
SVM Stacker	2.7137	0.0002	2.7136	2.7139	9
Simple	2.5216	0	2.5216	2.5216	8

TABLE XXXI
TEST RMSE: FTSE

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.3895	0.0022	2.3879	2.3911	1
ET	2.3982	0.0023	2.3966	2.3998	3
ADA	2.4066	0.0006	2.4062	2.4071	4
GB	2.3913	0.0023	2.3897	2.393	2
XGB	2.4361	0	2.4361	2.4361	6
Simple Stacker	2.6319	0.0057	2.6278	2.636	9
Network Stacker	2.5827	0.0183	2.5697	2.5958	8
SVM Stacker	2.5684	0.0002	2.5682	2.5685	7
Simple	2.4269	0	2.4269	2.4269	5

TABLE XXXII
TRAIN RMSE: N225

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	3.2519	0.0012	3.2511	3.2527	7
ET	3.2711	0.0012	3.2703	3.272	8
ADA	3.2114	0.0024	3.2097	3.2131	6
GB	3.1536	0.0005	3.1533	3.154	5
XGB	2.6783	0	2.6783	2.6783	3
Simple Stacker	2.4394	0.0009	2.4388	2.4401	1
Network Stacker	2.5031	0.0825	2.4441	2.5621	2
SVM Stacker	2.9765	0.0039	2.9737	2.9793	4
Simple	3.2947	0	3.2947	3.2947	9

Fig. 21. FTSE regression train results

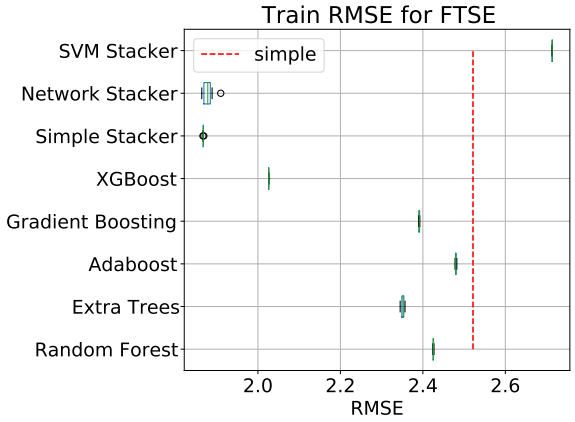


TABLE XXXIII
TEST RMSE: N225

	Mean	Std. dev	CI: lower	CI: upper	rank
RF	2.8954	0.0004	2.8951	2.8957	2
ET	2.894	0.0003	2.8938	2.8942	1
ADA	2.9007	0.0004	2.9004	2.901	4
GB	2.9131	0.0009	2.9125	2.9138	5
XGB	3.0535	0	3.0535	3.0535	6
Simple Stacker	3.3837	0.0053	3.3799	3.3875	9
Network Stacker	3.3597	0.1001	3.2881	3.4313	8
SVM Stacker	3.3085	0.0027	3.3066	3.3105	7
Simple	2.8971	0	2.8971	2.8971	3

Fig. 22. FTSE regression test results

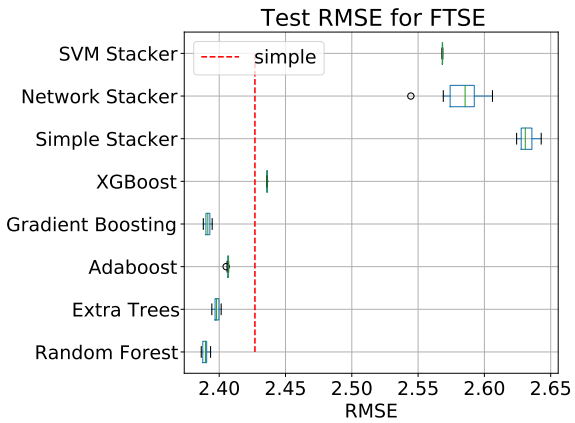


Fig. 23. N225 regression train results

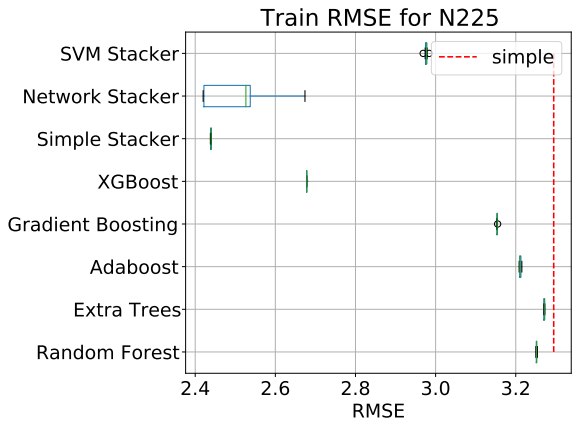
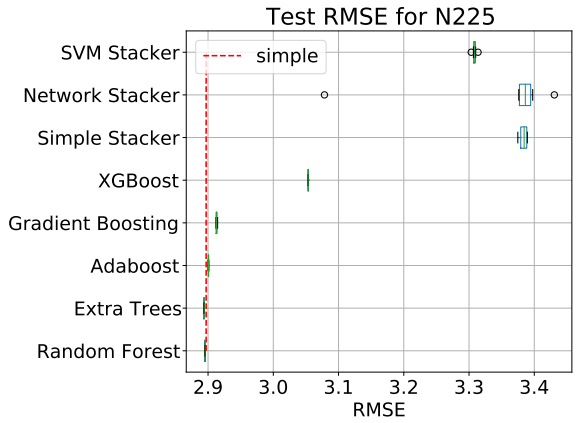


Fig. 24. N225 regression test results



REFERENCES

[1] C.-F. Tsai, Y.-C. Lin, D. C. Yen, and Y.-M. Chen, "Predicting stock returns by classifier ensembles," *Applied Soft Computing*, vol. 11, no. 2, pp. 2452–2459, 2011.

[2] M. Ballings, D. Van den Poel, N. Hespeels, and R. Gryp, "Evaluating multiple classifiers for stock price direction prediction," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7046–7056, 2015.

[3] B. Weng, L. Lu, X. Wang, F. M. Megahed, and W. Martinez, "Predicting short-term stock prices using ensemble methods and online data sources," *Expert Systems with Applications*, vol. 112, pp. 258–273, 2018.

[4] M. Jiang, J. Liu, L. Zhang, and C. Liu, "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*, vol. 541, p. 122272, 2020.

[5] I. E. Livieris, E. Pintelas, S. Stavroyiannis, and P. Pintelas, "Ensemble deep learning models for forecasting cryptocurrency time-series," *Algorithms*, vol. 13, no. 5, p. 121, 2020.

[6] "World indices." [Online]. Available: https://au.finance.yahoo.com/world-indices

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[10] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.