Comparative Analysis of Machine Learning Algorithms for Stock Price Forecasting on Big Data Platforms

Dr. Thallapally Sameera Pallavi

Freelance Faculty
Computer Science Department
Kakatiya University, Warangal,
Telangana-506009, India.
Mail ID: thallapally.sameerapallavi@gmail.com

Abstract:
There is mixed evidence that modern machine learning models are able to outperform naive benchmarks in predicting the price of the share. This study benchmarks ridge regression, random forest, gradient-boosted trees, multilayer perceptron, and support vector regression against a random-walk baseline for the next-day price prediction problem and quantifies the cost scale of training in the library of a single node and in Apache Spark MLlib. The data are 619,040 daily records for 505 S&P 500 stocks from 2013 to 2018. A total of nineteen technical features were calculated, the model was trained from 2013 to 2015 and the forecasts were evaluated sequentially on 138,536 stock-days from 2017 to early 2018. No algorithm outperformed the random walk. All of the models had negative out of sample r-squareds and no significant improvements were seen in Diebold-Mariano tests, with directional accuracy ranging from 49.7 percent to 50.2 percent. The random forest yielded a gross Sharpe ratio of 0.94 which was statistically significant from zero. For random forest training, Spark was up to 1.7 times faster than scikit-learn for the 1.38M rows, but for ridge regression and boosting training, Spark was significantly slower for one machine; and for kernel regression, it was impractical beyond tens of thousands of rows. The results confirm the need for evaluation on the basis of benchmarks and for matching the platform with the algorithm and the volume of data.

Keywords: stock price forecasting; machine learning; big data; Apache Spark; random forest; gradient boosting; efficient market hypothesis

1. Introduction

Prediction of a listed share price is one of the oldest and most controversial problems in quantitative finance. Investors, regulators and corporate treasurers all need to have some sense of what prices will be in the future, but the efficient market hypothesis states that prices should be unpredictable beyond a risk premium because they are already reflecting information that is available (Fama, 1970). That's something that has been proven time and time again through empirical evidence. Lo and MacKinlay (1988) had found that the random walk hypothesis could not be accepted for stock index returns that span weeks. Subsequent research by Gu et al. (2020) and by Krauss et al. (2017) found that well designed statistical models can yield small but economically significant signals from cross-sections of stocks. Rather, the real test for analysts will be which algorithms can recover a level of predictability when tested in an honest manner, outside of the sample and compared to a naïve benchmark.

There have been two developments which have affected the way this question is explored. The first one is the expansion of market data. A few hundred securities, watched day after day for a few years, will already produce hundreds of thousands of feature vectors per day (millions with every exchange), before the intraday, textual and other data are included. The second is the maturity of general purpose machine learning. Tree ensembles, kernel machines and neural networks are now included in standard libraries, and distributed engines like MapReduce and Apache Spark enable the same workflows to be executed across clusters of commodity machines (Dean & Ghemawat, 2008; Zaharia et al., 2016). Linear models, random forests and gradient-boosted trees (Meng et al., 2016) are distributed implementations provided by Spark's machine learning library, MLlib. These advances naturally raise the following questions: which algorithms offer the most accurate predictions and what is the computational expense?

Previous literature addresses some (but not all) of this question. Most of the studies examine only a few algorithms on a limited number of indexes or a few stocks, without providing a statistical comparison of the results against a random-walk baseline, and using a single workstation with no discussion of increasing training cost as a function of the size of the data set (Henrique et al., 2019; Shah et al., 2019). Extra difficulties are pointed out in the deep learning research of financial time series, including the fact that data sets, horizons, the way the model is validated and the metric used differ across papers (Sezer et al., 2020; Jiang, 2021).Such a reader is then left with little evidence that combines predictive accuracy, statistical testing and scalability within a single protocol if they wish to deploy on a distributed platform.

This paper fills that void by making a controlled comparison on a large, publicly accessible data set: 5 years of daily price data for the 505 constituents of the S&P 500 index. The study is guided by four research questions.

- RQ1. For a strictly chronological hold-out period, do widely-used machine learning algorithms outperform a random-walk benchmark in predicting the closing price the next day?
- RQ2. Are there any gains (or losses) over the benchmark that are still observed at formal test and when translated into a simple cross-sectional trading rule?
- RQ3. What are the best-performing ensembles using?
- RQ4. What is the relationship between training time and data size for a single node library versus apache spark mllib and does training platform affect the predictive accuracy?

The study has three contributions. The first is the reproducible benchmark, where the linear, kernel, tree-based and neural models all have one feature set, one split and one set of metrics. Second, it presents formal Diebold and Mariano (1995) tests and an economic back-test in addition to traditional error measures, thereby avoiding an undue interpretation of small differences in error measures, particularly in RMSE. Third, it compares the training time of the same data with different sizes of training set, using both scikit-learn (Pedregosa et al., 2011) and Spark MLlib, and therefore distinguishes the performance of an algorithm from the cost of a platform.

The rest of the paper is organized as follows: This chapter presents a review of the related work. The data and platform architecture are described in Section 3. A methodology is provided in
Section 4. The results are presented in Section 5, implications and limitations are discussed in Section 6 and conclusions are drawn in Section 7.

2. Literature Review

2.1 Market efficiency and the limits of predictability

The efficient market hypothesis forms the theoretical framework of any forecasting research, and it suggests three levels of integration of information in the price: weak, semi-strong and strong forms (Fama, 1970). Weak form efficiency suggests that market prices and volumes over the last few periods can't be exploited to gain systematic excess returns, the very same information basis upon which technical indicators rely. There is some evidence of lack of weak-form efficiency but it is limited. Lo and MacKinlay (1988) have determined that weekly returns on stock indices are positively autocorrelated and not completely random, and Welch and Goyal (2008) reported that many commonly-used stock index predictors underperform relative to the historical average when tested out of sample. The results are the reason why the naive benchmark should always be part of a careful forecasting study, because if the signal-to-noise ratio is low, a complicated model may well fit the noise in samples, and fail the test in the hold-out period.

2.2 Classical machine learning for price and direction forecasting

Early machine learning studies concentrated on neural networks and support vector machines applied to index data. Kim (2003) used support vector machines to predict the Korea Composite Stock Price Index and found them competitive with back-propagation networks, with performance sensitive to kernel and regularisation choices. Kara et al. (2011) compared neural networks and support vector machines for the direction of the Istanbul Stock Exchange National 100 index and reported that both exceeded chance, with the neural network slightly ahead. Patel et al. (2015) studied two stock indices and two individual stocks from India using technical indicators as inputs and compared artificial neural networks, support vector machines, random forests and naive Bayes. They showed that representing the indicators as trend-deterministic signals improved every classifier and that random forests performed particularly well in that representation. Ballings et al. (2015) extended the comparison to more than five thousand European companies and found that ensemble methods, especially random forests, produced the best area under the curve for one-year-ahead direction forecasts.

2.3 Ensembles and deep learning on broad cross-sections

Studies that use many stocks at once have changed the tone of the literature. Krauss et al. (2017) applied deep neural networks, gradient-boosted trees and random forests to S&P 500 constituents from 1992 to 2015 in a statistical arbitrage design and found that an equal-weighted ensemble produced large daily returns before transaction costs, although profitability weakened in the later part of the sample. Fisher and Krauss (2018) repeated the exercise using long short-term memory networks (Hochreiter & Schmidhuber, 1997) and discovered that these networks could outperform classifiers without memory, and with a decrease in returns over time. Gu et al. (2020) gave the best evidence for the monthly returns, by testing penalised linear models, trees and neural networks on thousands of US stocks and hundreds of characteristics. They had small out-of-sample R squared, < 1% per month in absolute size, but they were higher in the nonlinear model that includes interactions. Stacked autoencoders with long short-term memory (LSTM) networks have also been proposed for price series, such as stacked autoencoders with long short-term memory networks (Bao et al., 2017), or state-frequency memory networks (Zhang et al., 2017) and recurrent networks for intraday direction (Nelson et al., 2017). This fast growth is described in comprehensive reviews, which also highlight the lack of uniformity in the evaluation processes (Atsalakis & Valavanis, 2009; Henrique et al., 2019; Jiang, 2021; Sezer et al., 2020; Shah et al., 2019).

Today, gradient boosting (Friedman, 2001) and its scalable variations (XGBoost Chen & Guestrin, 2016; LightGBM Ke et al, 2017) are prevailing for tabular prediction tasks. Random forests (Breiman, 2001) are

still appealing due to their few tuning parameters and ability to provide variable-importance diagnostics. The two families are available in distributed fashion in Spark MLlib.

2.4 Big data platforms for financial analytics

The volume, velocity and variety of big data is applicable for market data as noted by Gandomi and Haider (2015). MapReduce has proposed a programming paradigm for fault-tolerant batch processing on clusters of computers (Dean & Ghemawat, 2008). Spark also implemented generalised as resilient distributed datasets, which store intermediate results in memory and are good for iterative algorithms like gradient boosting (Zaharia et al., 2012, 2016). MLlib exposes a common application programming interface for feature transformation and model training on top of this engine (Meng et al., 2016). Although finance is frequently cited as a motivating domain, published stock-forecasting studies seldom report how training time and accuracy behave as the data grow, which leaves practitioners to extrapolate from unrelated benchmarks.

2.5 Summary of prior evidence and research gap

Table 1 summarizes some representative studies. The following three observations are the motivation for the present design. There is general agreement that there is evidence of predictability for broad cross-sections and for nonlinear models, and little evidence for single series evaluated by raw error. Gains reported will be large in some periods and small in others, and will depend upon the transaction costs. Last, but not least, the issue of scalability is usually talked about qualitatively. In the present study, then, a wide cross-section was coupled with a random-walk benchmark and formal testing and explicit measures of scalability.

Table 1

Selected Prior Studies on Machine Learning for Stock Forecasting

Study	Data	Methods	Main finding
Kim (2003)	Korea Composite Stock Price Index, daily	SVM, back-propagation network, case-based reasoning	SVM competitive; sensitive to kernel and regularisation settings
Kara et al. (2011)	Istanbul Stock Exchange National 100 index, daily direction	Neural network, SVM	Both exceeded chance; neural network slightly ahead
Patel et al. (2015)	Two Indian indices and two stocks	ANN, SVM, random forest, naive Bayes	Trend-deterministic inputs helped all models; random forest strong
Ballings et al. (2015)	More than 5,000 European firms, one-year direction	Random forest, AdaBoost, kernel factory, neural network, logistic, SVM, k-NN	Ensembles, notably random forest, gave the highest AUC
Krauss et al. (2017)	S&P 500 constituents, 1992-2015	Deep network, gradient-boosted trees, random forest, ensemble	Large pre-cost daily returns; profitability weakened later

Study	Data	Methods	Main finding
Fischer and Krauss (2018)	S&P 500 constituents, 1992-2015	LSTM versus random forest, deep network, logistic regression	LSTM outperformed memory-free methods; later profits weaker
Bao et al. (2017)	Six stock indices	Stacked autoencoders with LSTM	Outperformed benchmark deep and recurrent models
Gu et al. (2020)	US stocks, monthly, 1957-2016	Penalised regression, trees, neural networks	Nonlinear models gave the largest, though small, out-of-sample R-squared

3. Data and Platform Architecture

3.1 Data set

The empirical analysis is based on the publicly available file of stock data for the S&P 500 from Nugent (2018), which includes daily closing price, trading volume, low and high prices for 505 tickers over the period 8 February 2013–7 February 2018. There are 1,259 trading days in the file and 619,040 stock-day records. Since the file is publicly available, results can be compared to future work done on the same records. The prices are as presented in the source data and the study does not re-adjust the prices for corporate actions, which is recognised as a limitation in Section 6. When performing a quality audit, there were no duplicated ticker-date pairs and 27 missing cells (11 opening prices, 8 high and 8 low prices). 11 records were removed because they had missing data, resulting in 619,029 records. The fields for "Price" were all strictly positive after cleaning, with the minimum volume being 101 shares.

Descriptive statistics provide a sense of the diversity that is crucial to the allure and challenge of cross-sectional pooling. The average closing price is approximately 83 USD with a standard deviation of approximately 97 USD, and a maximum price of over 2,000 USD. The trading volume is highly skewed with the median trading volume being about 2.1 million shares per day, and the mean being 4.3 million shares per day. The mean daily log return is 0.04 percent and its standard deviation is 1.6 percent. Because price-level error measures are dominated by high-priced stocks, the evaluation in Section 4 includes scale-free return-based measures as well.

3.2 Chronological partition

All models are trained on the past and tested on the future. Observations are split by date into a training period (11 March 2013 to 31 December 2015), a validation year (2016) and an untouched test period (3 January 2017 to 6 February 2018), as summarised in Table 2. Random shuffling is not used, because it would leak information across time and inflate accuracy (Bergmeir & Benítez, 2012). Figure 1 shows the equal-weighted cumulative return of the 505 stocks and the three periods. The test period is a strong bull market with unusually low volatility until a sharp sell-off in early February 2018, a feature that favours long-biased rules and should be remembered when interpreting the results.

Table 2

Data Set Description and Chronological Partition

Item	Period or description	Records
Source	Nugent (2018), daily OHLCV, 505 tickers, 1,259 trading days	619,040
After cleaning	Removal of records with missing prices (27 missing cells)	619,029
Modelling records	After rolling-window warm-up and removal of final day	608,424
Training	2013-03-11 to 2015-12-31	344,648
Validation (held out)	2016-01-04 to 2016-12-30	125,240
Test	2017-01-03 to 2018-02-06	138,536

Figure 1

Equal-Weighted Cumulative Return of 505 S&P 500 Stocks and the Chronological Data Partition



Note. Author's computation using data from Nugent (2018).

3.3 Platform architecture

Figure 2 depicts the pipeline. Raw files are ingested, stored as delimited files, transformed into features, used to train models, and finally evaluated on the hold-out period. On a production cluster the storage step would use a columnar format on a distributed file system, but here local files are used. The computation layer is implemented twice. In the single-node configuration, scikit-learn (Pedregosa et al., 2011) trains models in the memory of one machine, with tree ensembles using both available processor cores. In the distributed configuration, Spark MLlib (Meng et al., 2016) reads the same files into a DataFrame, partitions rows across executor threads and trains the same model families with data-parallel algorithms. All experiments ran on one machine with two processor cores and 7 GB of memory, with Spark in local mode using two worker threads. This arrangement isolates the software overhead and algorithmic behaviour of the engine, but it is not a multi-node cluster, and Section 6 discusses what this implies for generalisation.

Figure 2

Big Data Pipeline for Stock Price Forecasting Used in This Study



Note. Author's computation using data from Nugent (2018).

4. Methodology

4.1 Forecasting target

The goal is to forecast the next trading day's closing price for each stock. Directly regressing price levels on past prices invites spurious fit because levels are non-stationary and differ by orders of magnitude across tickers. The models therefore forecast the next-day log return and the price forecast is reconstructed from it:

$r(t+1) = \ln[\, P(t+1) / P(t) \,], \quad \hat{P}(t+1) = P(t) \times \exp[\, \hat{r}(t+1) \,]$

where P(t) is the closing price on day t. A forecast of zero return reproduces the random-walk benchmark, which predicts that tomorrow's close equals today's close. Because the benchmark is nested in every model, any improvement can be attributed to information the model extracted from the features.

4.2 Feature engineering

Nineteen features, listed in Table 3, were computed separately for each stock using only information available at the close of day t. They comprise lagged returns over one to five days and cumulative returns over 5, 10 and 20 days; ratios of the closing price to its 5-, 10- and 20-day simple moving averages; the moving average convergence divergence measure scaled by price; the 14-day relative strength index (Wilder, 1978); realised volatility over 5 and 20 days; the intraday high-low range; the open-to-close gap; a volume ratio comparing current volume with its 20-day average; and the day of the week. These are standard technical indicators described in practitioner texts (Murphy, 1999) and widely used in the studies reviewed above (Patel et al., 2015). Records with incomplete rolling windows were dropped, which yields 608,424 modelling records. To limit the influence of data errors and extreme events such as unadjusted splits, features were winsorised at the 0.1st and 99.9th percentiles of the training period, and the training and validation targets were clipped at the same percentiles. The test targets were never clipped, so reported errors reflect the actual extremes of the hold-out period.

Table 3

Input Features Computed for Each Stock-Day

No.	Feature	Definition	Group
1-5	ret_lag1 ... ret_lag5	Daily log returns on days t, t-1, ..., t-4	Momentum
6-8	ret_5d, ret_10d, ret_20d	Cumulative log return over 5, 10 and 20 days	Momentum
9-11	sma5/10/20_ratio	Closing price divided by 5-, 10- and 20-day simple moving average, minus one	Trend

No.	Feature	Definition	Group
12	macd	(12-day EMA minus 26-day EMA of close) divided by close	Trend
13	rsi14	Relative strength index, 14-day exponential smoothing (Wilder, 1978)	Oscillator
14-15	vol5, vol20	Standard deviation of daily log returns over 5 and 20 days	Volatility
16	hl_range	(High minus low) divided by close	Volatility
17	oc_gap	(Close minus open) divided by open	Intraday
18	vol_ratio	Log of volume divided by its 20-day average	Liquidity
19	dow	Day of week (0 = Monday)	Calendar

4.3 Algorithms

Six forecasting approaches were compared, with configurations in Table 4. The random-walk benchmark forecasts a zero return. Ridge regression is a linear model with an L2 penalty and serves as a transparent baseline (Hastie et al., 2009). Random forests average many decorrelated regression trees grown on bootstrap samples (Breiman, 2001). Gradient-boosted trees build trees sequentially, each fitted to the residual gradient of the current ensemble (Friedman, 2001); the scikit-learn histogram-based implementation was used. The multilayer perceptron has two hidden layers of 64 and 32 units with early stopping and is a universal approximator in principle (Hornik et al., 1989). Support vector regression with a radial basis kernel (Cortes & Vapnik, 1995; Kim, 2003) has training cost that grows between quadratically and cubically in the number of samples, so it was trained on a random subsample of 20,000 training rows, which is itself a finding about scalability. Ridge, the perceptron and the support vector model used features standardised with training-period means and standard deviations; tree models used raw features.

Hyperparameters were fixed in advance at conventional values rather than tuned, so that the same settings could be used in scikit-learn and Spark and so that the comparison reflects algorithm families rather than search effort. The validation year was held out but not used for tuning, and this restriction is discussed among the limitations.

Table 4

Model Configurations

Model	Library	Key settings	Training rows
Random walk	-	Forecast log return = 0 (price unchanged)	-
Ridge regression	scikit-learn / MLlib	L2 penalty (alpha = 10), standardised inputs	344,648
Random forest	scikit-learn / MLlib	50 trees, depth 10, min. 50 per leaf, 50% of features per split	344,648

Model	Library	Key settings	Training rows
Gradient boosting	scikit-learn (histogram) / MLlib	50 iterations, depth 5, learning rate 0.1, min. 50 per leaf	344,648
MLP	scikit-learn	Hidden layers 64-32, ReLU, L2 = 0.001, early stopping	344,648
SVR	scikit-learn	RBF kernel, C = 1, epsilon = 0.001, standardised inputs	20,000 (random)

4.4 Distributed implementation and scalability design

Spark MLlib provides linear regression, random forest regression and gradient-boosted tree regression. They were configured to match the scikit-learn models: 50 trees of depth 10 with at least 50 instances per node and half of the features considered at each split for the forest, and 50 boosting iterations of depth-5 trees with a step size of 0.1 for boosting. Two differences are inherent to the engines. MLlib discretises continuous features into at most 32 bins before searching for splits, which reduces communication in distributed settings, whereas the scikit-learn random forest evaluates exact thresholds. Spark also incurs scheduling and serialisation overhead that is amortised only on larger data.

To study scalability, training sets of 10, 25, 50 and 100 percent of the training period were drawn at random. Because the real data are only about 345,000 rows, two larger sets were built by replicating the training data twice and four times, adding zero-mean Gaussian noise with a standard deviation of one percent of each feature's standard deviation to the copies. These enlarged sets, of about 689,000 and 1.38 million rows, are used only for timing and never for accuracy. Each timing is a single fit measured after a warm-up run, with the data cached in memory; times exclude data loading and JVM start-up.

4.5 Evaluation protocol

Accuracy is evaluated on the 138,536 test records. At the price level, the root mean squared error (RMSE), mean absolute error (MAE) and mean absolute percentage error (MAPE) of the next-day close are reported; because MAPE has known weaknesses, it is shown alongside scale-free alternatives (Hyndman & Koehler, 2006). At the return level, the study reports the RMSE of the return forecast, the out-of-sample R-squared measured against the zero-return benchmark,

$R^2(OOS) = 1 - \Sigma [r - \hat{r}]^2 / \Sigma r^2$,

and directional accuracy, the share of records where the sign of the forecast matches the realised move. A negative R^2(OOS) means a model is worse than the random walk. Following Diebold and Mariano (1995), the daily cross-sectional mean of squared return errors is computed for the benchmark and for each model, and the average loss differential over the 276 test days is tested with a Newey and West (1987) variance estimator using five lags. Finally, a cross-sectional back-test buys the decile of stocks with the highest forecasts and sells the decile with the lowest each day, holding for one day with equal weights; annualised return, volatility and the Sharpe ratio are reported before transaction costs. All random seeds were fixed at 42 for reproducibility.

5. Results and Discussion

5.1 Forecast accuracy against the random-walk benchmark

Table 5 and Figure 3 report accuracy on the 138,536 test records. The central result is that none of the five algorithms improves materially on the random walk. At the return level, which is scale-free and weights every stock equally, the benchmark attains the lowest RMSE of 1.4810 percent. Ridge regression, the random forest, gradient boosting and the multilayer perceptron are all marginally worse, at 1.4836 to 1.4847 percent, and the support vector regression is clearly worse at 1.6847 percent. The out-of-sample R-squared against the zero-return forecast is therefore negative for every model, ranging from -0.34 percent for the random forest to -0.50 percent for the perceptron and -29.4 percent for the support vector model, which was trained on only 20,000 rows and overfitted the noise in that sample.

Directional accuracy tells the same story. The four main models call the sign of the next-day move correctly in 49.8 to 50.2 percent of stock-days, indistinguishable from a coin toss. A rule that always predicted an up move would have scored 52.7 percent, because 52.7 percent of next-day returns in the bull-market test period were positive. Monthly directional accuracy of the gradient-boosting forecasts stayed between 49 and 51.5 percent in twelve of the fourteen test months, with deviations only in August 2017 (46.4 percent) and the first three trading days of February 2018 (55.9 percent). The deviation in February 2018 rests on a handful of days and should not be read as skill.

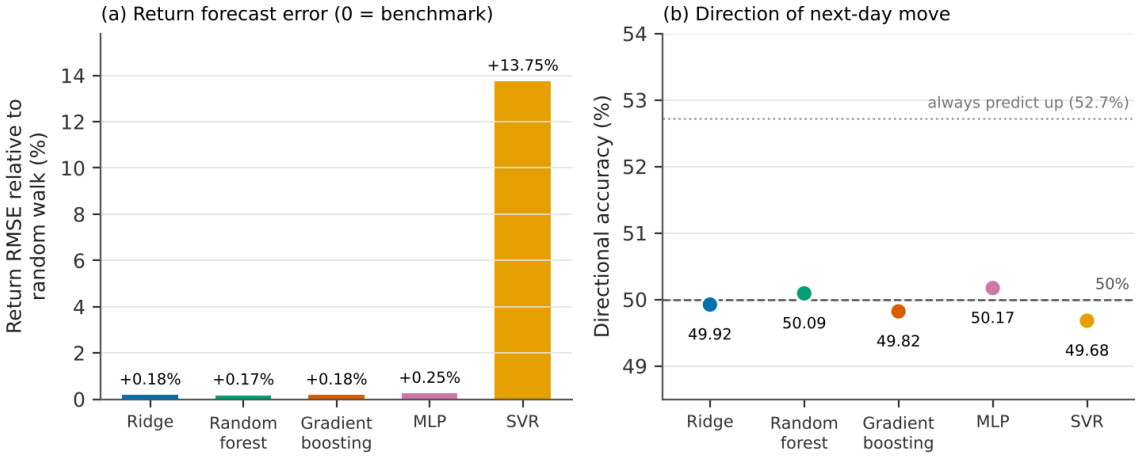
Table 5

Test-Period Forecast Accuracy (3 January 2017 to 6 February 2018; 138,536 Stock-Days)

Model	RMSE ($)	MAE ($)	MAPE (%)	Return RMSE (%)	R^2 OOS (%)	Dir. acc. (%)
Random walk (benchmark)	2.1814	0.8964	0.9464	1.4810	0.000	-
Ridge regression	2.1835	0.8984	0.9489	1.4837	-0.360	49.92
Random forest	2.1789	0.8974	0.9490	1.4836	-0.343	50.09
Gradient boosting	2.1777	0.8982	0.9500	1.4837	-0.358	49.82
MLP (64-32)	2.1749	0.8987	0.9516	1.4847	-0.496	50.17
SVR (20k sample)	2.4602	1.0435	1.1082	1.6847	-29.396	49.68

Figure 3

Test-Period Accuracy by Model: (a) RMSE of the Next-Day Log Return and (b) Directional Accuracy



Note. Author's computation using data from Nugent (2018).

At the price level the picture appears slightly different, and the difference deserves explanation. The perceptron, gradient boosting and random forest show RMSEs of 2.1749, 2.1777 and 2.1789 dollars, against 2.1814 dollars for the benchmark, a reduction of 0.1 to 0.3 percent. Mean absolute error and mean absolute percentage error, however, are lowest for the benchmark. Price-level RMSE squares errors measured in dollars, and dollar errors scale with the price. Six of the 505 stocks, those with average test-period closing prices above 500 dollars, account for 53 percent of the sum of squared closing prices, so their errors carry about half of the weight and differences of this size can come from a handful of names. Following the recommendation to look beyond a single scale-dependent measure (Hyndman & Koehler, 2006), the study treats the price-level gap as too small to be evidence of forecasting skill, and the formal tests below support that reading. Taken together, the results answer RQ1 negatively: on this data set and horizon, none of the compared algorithms beats the random walk.

The scale of the negative R-squared values is worth stating. Gu et al. (2020) report positive but tiny out-of-sample R-squared values for monthly returns, below one percent. A daily horizon is noisier than a monthly one relative to any predictable component, so there is little reason to expect larger positive values here, and the figures of -0.3 to -0.5 percent indicate that estimation error in the fitted models slightly exceeds whatever signal they captured.

5.2 Statistical significance

Table 6 reports Diebold and Mariano (1995) tests of equal predictive accuracy against the benchmark. All loss differentials are negative, meaning that the benchmark has the smaller squared error on average. The differences are statistically insignificant for the random forest ($p = 0.140$), gradient boosting ($p = 0.241$) and the perceptron ($p = 0.205$), so these models are statistically indistinguishable from the random walk rather than better than it. Ridge regression is significantly worse at the 5 percent level ($p = 0.039$), and the support vector regression is far worse ($p < 0.001$). The tests use daily cross-sectional averages, which account for the strong contemporaneous correlation of errors across stocks. Treating the 138,536 records as independent would have produced misleadingly small standard errors.

Five models are compared with the same benchmark, so some caution about multiple testing is appropriate. The direction of the evidence makes this less of a concern: the only significant differences, for ridge regression and the support vector model, indicate worse performance than the benchmark, and an adjustment for multiple

comparisons could only weaken them. The conclusion that no model is better than the random walk therefore does not depend on the number of comparisons made.

Table 6

Diebold-Mariano Tests Against the Random Walk (276 Test Days, Newey-West Lag 5)

Model	Mean loss differential ($\times 10^{-7}$)	DM statistic	p value
Ridge regression	-7.90	-2.06	0.039
Random forest	-7.54	-1.47	0.140
Gradient boosting	-7.88	-1.17	0.241
MLP (64-32)	-10.93	-1.27	0.205
SVR (20k sample)	-644.33	-18.32	<0.001

Note. Loss differential = benchmark squared error minus model squared error, averaged across stocks each day. Negative values mean the model is less accurate than the benchmark.

5.3 Economic significance

Statistical equivalence to the benchmark does not rule out a profitable cross-sectional ordering of stocks, because a model can rank stocks usefully even when its forecasts are poorly calibrated. Table 7 therefore reports the decile long-short back-test. The random forest earns a gross mean of 3.4 basis points per day, an annualised return of 8.6 percent and a Sharpe ratio of 0.94, with profitable days on 54.3 percent of the 276 test days. Gradient boosting earns a smaller positive return, while ridge regression, the perceptron and the support vector model lose money. Even the best result is not statistically distinguishable from zero: its t-statistic is about 1.0 over a single year. It is also measured before transaction costs. If the portfolio were completely replaced each day, a one-way trading cost of under one basis point would remove the random forest's gross profit. The dispersion across models is itself informative, because five similar-looking algorithms produced Sharpe ratios from -1.05 to 0.94 in the same year. This pattern is as expected when the actual ranking is predominantly noise and for any given ensemble, a successful performance on one model should not be considered as a true sign of success and benefits in general. This is in answer to RQ2 that no model is able to guarantee an economic gain, and a successful performance on one model is not a generalised sign of success and benefits for the ensemble.

Table 7

Decile Long-Short Back-Test, Gross of Transaction Costs

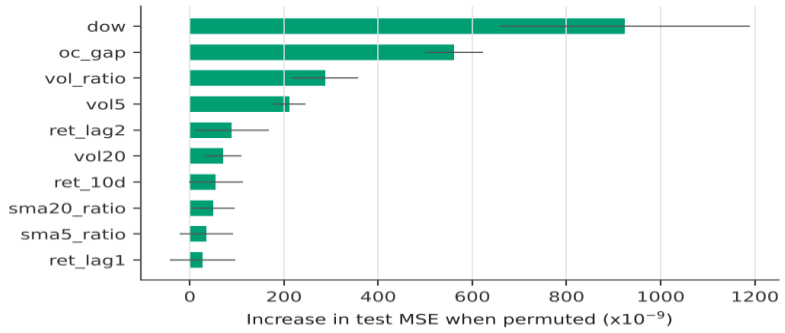
Model	Mean daily (bp)	Ann. return (%)	Ann. vol. (%)	Sharpe	t-stat	Win days (%)
Ridge regression	-4.58	-11.53	11.00	-1.05	-1.10	47.8
Random forest	3.39	8.55	9.13	0.94	0.98	54.3
Gradient boosting	1.19	3.00	8.72	0.34	0.36	50.0
MLP (64-32)	-0.97	-2.45	8.57	-0.29	-0.30	48.6
SVR (20k sample)	-2.38	-5.99	6.37	-0.94	-0.98	46.0

5.4 Variable importance

Figure 4 shows permutation importance for the random forest on a random sample of 15,000 test stock-days: how much the mean squared error rises when one feature is shuffled. The day of the week ranks first, followed by the open-to-close gap, the volume ratio and short-term volatility. Lagged returns and trend ratios rank lower. All values are small relative to the forest's overall mean squared error of about 2.2 x 10^-4; even the largest increase is about 0.4 percent of it. Two readings are consistent with this evidence. It does not make use of features that describe the market's calendar and liquidity characteristics as much as it does of the stock-specific past return, and it adapts to the average day-of-week patterns from the training years that do not go over into net gains as forecast in the test year, since the model still performs worse than a zero-forecast. Permutation importance measures how much a fitted model depends on a variable, not whether that dependence generalises, so these results answer RQ3 only descriptively: the forest leans on calendar, intraday and volatility information, and none of it is converted into out-of-sample skill.

Figure 4

Permutation Importance of the Ten Most Influential Features in the Random Forest



Note. Author's computation using data from Nugent (2018).

5.5 An illustrative stock

Figure 5 plots the realised closing price of Apple Inc. together with the gradient-boosting and ridge forecasts for the test period. The forecast lines are visually indistinguishable from the realised series, and that closeness is exactly what a one-day-ahead random walk would show. A model whose forecast is today's price multiplied by a number close to one will always appear accurate on a price chart, which is why charts of this kind are poor evidence of forecasting ability and why the evaluation here relies on returns, formal tests and a benchmark. The close-up panel for the last forty days shows the forecasts lagging the realised series by one day, the familiar signature of a naive forecaster.

Figure 5

Actual and Forecast Closing Prices for Apple Inc. (AAPL) in the Test Period



Note. Author's computation using data from Nugent (2018).

5.6 Scalability of training

Table 8 and Figure 6 compare the time taken in the training phase with the same data. It's a very different picture for each algorithm, and one platform doesn't win in all markets.

At both scales, scikit-learn is 9 to 22 times faster than Spark for ridge regression (0.06 seconds running on 34,465 rows, 0.62 seconds on 1.38 million rows, compared to 0.63 and 8.8 seconds respectively). The Spark time increased from 345,000 to 689,000 rows, with no obvious explanation in terms of the algorithmic cost of the job, and likely due to memory or local mode partitioning effects.

Because every ridge fit finished within ten seconds, it is not pursued further.

The random forest shows a crossover. Spark is slower on the smallest set (12.8 against 6.3 seconds), roughly equal at 86,162 rows (19.2 against 18.3 seconds), and faster from about 170,000 rows. On the full training period, it needs 69.0 seconds against 92.4 seconds, and on 1.38 million rows, 251 seconds against 422 seconds, a speed-up of 1.7 times. The log-log slope of training time on rows is 1.14 for scikit-learn, slightly super-linear, and 0.81 for Spark, which indicates that fixed start-up costs are being amortised. Both engines used the same two processor cores, so the advantage cannot be attributed to adding hardware. It plausibly reflects Spark's coarse discretisation of features into 32 bins compared with exact threshold search in the scikit-learn forest, though this explanation was not tested separately.

Gradient boosting reverses the ranking. The scikit-learn histogram implementation needs only 0.24 seconds on the smallest set and 5.9 seconds on the largest. Spark needs 25.0 and 145 seconds, between 24 and 104 times longer. Spark's curve is much flatter, with a slope of 0.48 against 0.87, which is the signature of a high fixed cost per fit. Boosting trains trees one after another, and each tree requires several scheduled passes over the data, a cost that is hard to amortise on a single machine. Whether the balance shifts on data that do not fit into one machine's memory cannot be answered here.

Table 8

Training Time (Seconds) by Training-Set Size: scikit-learn Versus Spark MLlib

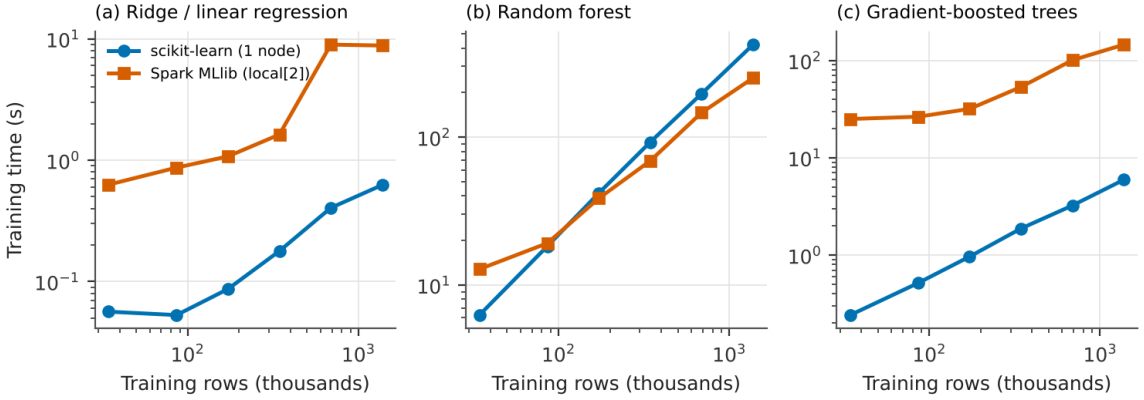
Rows	Ridge sklearn	Ridge Spark	RF sklearn	RF Spark	GBT sklearn	GBT Spark
34,465	0.06	0.63	6.26	12.79	0.24	24.96
86,162	0.05	0.87	18.30	19.17	0.52	26.34
172,324	0.09	1.07	41.76	38.52	0.96	31.70

Rows	Ridge sklearn	Ridge Spark	RF sklearn	RF Spark	GBT sklearn	GBT Spark
344,648	0.18	1.63	92.41	69.02	1.87	53.29
689,296	0.40	8.94	196.4	146.3	3.23	100.4
1,378,592	0.62	8.79	421.6	250.8	5.93	145.0

Note. Rows above 344,648 are replicated training data used for timing only. Spark ran in local mode with two threads on one machine.

Figure 6

Training Time Versus Number of Training Rows (Log-Log Axes): scikit-learn Versus Spark MLlib

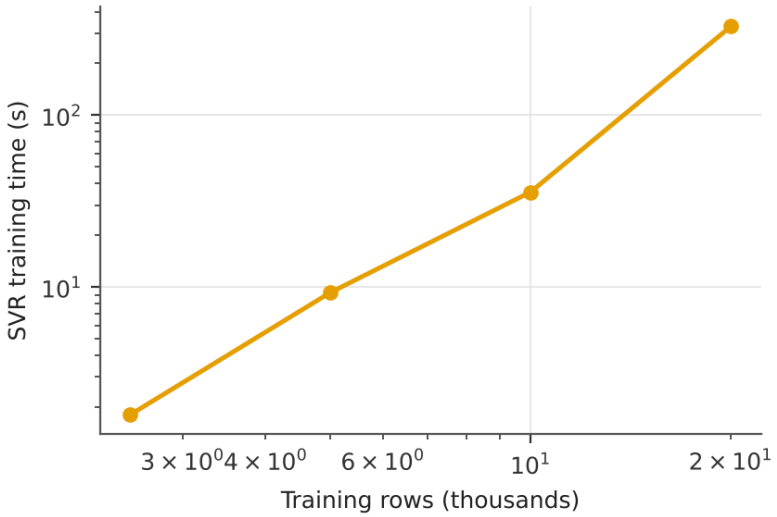


Note. Author's computation using data from Nugent (2018).

Kernel methods face a different barrier. Figure 7 shows that support vector regression took 1.8 seconds on 2,500 rows and 328 seconds on 20,000 rows. An eightfold increase in rows raised time 182-fold, a slope of about 2.5 on log-log axes. If growth continued at a quadratic rate, a fit on the full 344,648 training rows would take about 27 hours, and at a cubic rate about 19 days. These are illustrations rather than measurements, but they show why the model was confined to a subsample and why it is unsuited to this scale. The first part of RQ 4 was then answered as follows: for tree ensembles, the time increases linearly, and for kernel methods it increases faster; and the benefit of using Spark will depend on the algorithm and on the minimum size of the data.

Figure 7

Training Time of RBF Support Vector Regression Versus Sample Size (Log-Log Axes)



Note. Author's computation using data from Nugent (2018).

5.7 Does the platform change accuracy?

Table 9 compares test-period accuracy for the three model families implemented on both platforms. Ridge regression and the random forest give practically identical results. Return RMSE is 1.4837 percent (scikit-learn) against 1.4836 percent (Spark) for ridge, and 1.4836 percent against 1.4842 percent for the forest, with out-of-sample R-squared between -0.34 and -0.43 percent. Gradient boosting differs more: the Spark version records a return RMSE of 1.4913 percent and an R-squared of -1.40 percent, against 1.4837 percent and -0.36 percent in scikit-learn, and a directional accuracy of 49.5 percent. The differences are most likely implementation details, for example, the number of feature bins, since the scikit-learn histogram booster uses up to 255 bins by default and MLlib was set to 32, together with regularisation and tie-breaking rules, rather than an effect of distribution as such. Neither implementation beats the random walk, so the substantive conclusion is the same on both platforms. This answers the second half of RQ4: the platform changes the cost of training far more than the quality of the forecast, although the match is not exact for boosted trees.

Table 9

Accuracy and Training Time by Platform (Full Training Set, 344,648 Rows)

Model	Platform	RMSE ($)	Return RMSE (%)	R^2 OOS (%)	Dir. acc. (%)	Time (s)
Ridge regression	scikit-learn	2.1835	1.4837	-0.360	49.92	0.1
Ridge regression	Spark MLlib	2.1837	1.4836	-0.349	49.89	3.3
Random forest	scikit-learn	2.1789	1.4836	-0.343	50.09	88.8
Random forest	Spark MLlib	2.1785	1.4842	-0.432	50.18	69.0
Gradient boosting	scikit-learn	2.1777	1.4837	-0.358	49.82	1.6
Gradient boosting	Spark MLlib	2.1831	1.4913	-1.396	49.50	51.7

6. Discussion, Implications and Limitations

6.1 Interpretation

For those readers who are familiar with the literature on market efficiency, the surprise of no algorithm outperforming the random walk should not come as a surprise. Weak-form efficiency implies exactly this result at a one-day time horizon, when the features are only constructed from past prices and volumes (Fama, 1970). The frequency with which seemingly successful predictors fail out-of-sample has been recorded by Welch and Goyal (2008, and the best nonlinear predictors explain well less than one percent of the variation in monthly returns, as found by gu et al, 2020. These out-of-sample Rs2s are negative, which is how overfitting to a low signal-to-noise problem appears. Even more complicated models failed to lift the forecasts: the perceptron, forest, and boosting did not improve over the benchmark, and the most flexible kernel model that was trained on a small sample was significantly poorer.

The contrast with other findings of Krauss et al. (2017) and Fischer and Krauss (2018) that substantial pre-cost profits can be achieved on S&P 500 constituents needs to be taken with a pinch of salt. The studies included in those covered the period 1992–2015, employed a trading design that incentivizes and encourages cross-sectional ranking, and in their own words found that profitability declined over time. The present test year is a short one, preceded by an unusually quiet volatility environment, and then followed by an unusually short one in which costs were not an issue, so the one positive back-test and the random forest's Sharpe ratio of 0.94 is hard to separate from luck. Those results are neither confirmed nor contradicted by the results. They demonstrate that conclusions are related to the time, the horizon and the benchmark.

6.2 Implications for practice

Four recommendations follow. First, any forecasting study ought to present a random walk model as a benchmark, and a formal test against the benchmark. The price charts shown in Figure 5 make any model look good. Secondly, dollar-based errors should be accompanied by scale-free return measures and/or cross-sectional aggregation, as a handful of expensive stocks can drive price-level RMSE. Third, the platform chosen should not necessarily be distributed, as it should be selected to meet the requirements of the data and algorithm. A single-node trained gradient boosting and ridge regression were faster than Spark at all sizes up to 1.38 million rows on this workload, and Spark had a small advantage for random forests starting around 170K rows. Fourthly, it is not advisable to use kernel methods at this scale unless approximated. Of course, practitioners should also report on timing ratios and hardware, since absolute times don't carry over to other environments! A team that has data stored in memory does not lose much by using one node, and for Spark, fault tolerance and integration requirements weren't factors in this experiment; neither was data volume.

6.3 Limitations

The findings are subject to a number of cautions. The data is from 1 market, daily frequency and 5 years – survivorship bias is possible, as the file is limited to the 505 constituents that existed in early 2018. There have been no price adjustments for corporate actions but winsorisation restricted the impact of extreme values. The formal tests did not obviously reverse the conclusion, although the hyperparameters were preset and the validation year was not used for tuning, so improvements to the models due to better hyperparameter tuning may help close the gaps. No recurrent architectures were implemented (such as long short-term memory networks), and a subsample was employed in support vector regression. The Spark experiments were conducted in local mode with a single two-core machine, and thus do not involve network communication, data shuffling or fault tolerance of a real cluster. Each of the two larger scalability sets was replicated with

noise and were used only for timing; therefore, the differences of a few seconds should not be over-read. Each of the timing was a single run. The back-test does not take into account transaction costs or market impact.
Future work could include an extension of the comparison to sequence models, intraday and text-based features, walk-forward tuning, a larger data set than memory to be used in a multi-node cluster scalability study, and cost-aware trading rules.

6.4 Availability of data and reproducibility

Data is freely available from its compiler (Nugent, 2018). Other researchers can reproduce the experiments as the preparation steps, feature definitions, hyperparameters, split dates and random seeds are reported in Sections 3 and 4 and the models are based on publicly available libraries. The timings will vary with other hardware and readers should compare ratios between platforms and not seconds.

7. Conclusion

This paper has compared five machine learning algorithms to a random-walk benchmark on forecasting the next day's price of 505 constituents of the S&P 500, and has compared the cost of training these algorithms on scikit-learn and Apache Spark's MLlib. The study has four conclusions.
The first part is that none of the algorithms predict better than the random walk given a strictly chronological test of the 138536 stock-days. All models had negative out of sample R-squared values, while directional accuracy was generally around 50 percent, and Diebold-Mariano tests did not produce any statistically significant models. Secondly, economic performance was unpredictable. The investment strategy based on a decile random forest generated a gross Sharpe ratio of 0.94, although it was not distinguishable from zero every year and would not withstand realistic trading costs, and other models resulted in negative returns. Third, calendar, intraday and volatility variables were the primary drivers of the forest, but not to the extent of being picked up as out-of-sample skill.
Fourthly, the result of platform comparison indicates that the computation cost varies more than the accuracy. Spark had a much faster run time than scikit-learn for random forests above approximately 170,000 rows, up to 1.7 times faster at 1.38 million rows, but much slower when it came to ridge regression and gradient boosting on this single machine, and beyond a few tens of thousands of rows, kernel regression was too slow to be practical. The quality of the forecasts was comparable between platforms for linear models and forests, and lower for Spark's boosted trees.
The key takeaway from the practical lesson is to first set a target and the statistics before complexifying models or infrastructures, and to select the computational platform based on the algorithm and the amount of data. The results should be seen as a baseline that others can use to replicate or compare with richer data sets, more complex architectures and real cluster deployments.

REFERENCES:

1. Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques – Part II: Soft computing methods. *Expert Systems with Applications, 36*(3), 5932–5941. https://doi.org/10.1016/j.eswa.2008.07.006
2. Ballings, M., Van den Poel, D., Hespeels, N., & Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications, 42*(20), 7046–7056. https://doi.org/10.1016/j.eswa.2015.05.013
3. Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE, 12*(7), Article e0180944. https://doi.org/10.1371/journal.pone.0180944

4. Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences, 191*, 192–213. https://doi.org/10.1016/j.ins.2011.12.028
5. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324
6. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785
7. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273–297. https://doi.org/10.1007/BF00994018
8. Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113. https://doi.org/10.1145/1327452.1327492
9. Diebold, F. X., & Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business & Economic Statistics, 13*(3), 253–263. https://doi.org/10.1080/07350015.1995.10524599
10. Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance, 25*(2), 383–417. https://doi.org/10.2307/2325486
11. Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research, 270*(2), 654–669. https://doi.org/10.1016/j.ejor.2017.11.054
12. Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics, 29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451
13. Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management, 35*(2), 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007
14. Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies, 33*(5), 2223–2273. https://doi.org/10.1093/rfs/hhaa009
15. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7
16. Henrique, B. M., Sobreiro, V. A., & Kimura, H. (2019). Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications, 124*, 226–251. https://doi.org/10.1016/j.eswa.2019.01.012
17. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
18. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8
19. Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting, 22*(4), 679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001
20. Jiang, W. (2021). Applications of deep learning in stock market prediction: Recent progress. *Expert Systems with Applications, 184*, Article 115537. https://doi.org/10.1016/j.eswa.2021.115537
21. Kara, Y., Boyacioglu, M. A., & Baykan, Ö. K. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert Systems with Applications, 38*(5), 5311–5319. https://doi.org/10.1016/j.eswa.2010.10.027
22. 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 30* (pp. 3146–3154). Curran Associates.

23. Kim, K.-J. (2003). Financial time series forecasting using support vector machines. *Neurocomputing, 55*(1–2), 307–319. https://doi.org/10.1016/S0925-2312(03)00372-2
24. Krauss, C., Do, X. A., & Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research, 259*(2), 689–702. https://doi.org/10.1016/j.ejor.2016.10.031
25. Lo, A. W., & MacKinlay, A. C. (1988). Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies, 1*(1), 41–66. https://doi.org/10.1093/rfs/1.1.41
26. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D. B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., & Talwalkar, A. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research, 17*(34), 1–7. https://jmlr.org/papers/v17/15-237.html
27. Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. New York Institute of Finance.
28. Nelson, D. M. Q., Pereira, A. C. M., & de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 1419–1426). IEEE. https://doi.org/10.1109/IJCNN.2017.7966019
29. Newey, W. K., & West, K. D. (1987). A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica, 55*(3), 703–708. https://doi.org/10.2307/1913610
30. Nugent, C. (2018). *S&P 500 stock data* [Data set]. Kaggle. https://www.kaggle.com/datasets/camnugent/sandp500
31. Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications, 42*(1), 259–268. https://doi.org/10.1016/j.eswa.2014.07.040
32. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. https://jmlr.org/papers/v12/pedregosa11a.html
33. Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing, 90*, Article 106181. https://doi.org/10.1016/j.asoc.2020.106181
34. Shah, D., Isah, H., & Zulkernine, F. (2019). Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies, 7*(2), Article 26. https://doi.org/10.3390/ijfs7020026
35. Welch, I., & Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies, 21*(4), 1455–1508. https://doi.org/10.1093/rfs/hhm014
36. Wilder, J. W., Jr. (1978). *New concepts in technical trading systems*. Trend Research.
37. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation* (pp. 15–28). USENIX Association.
38. Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016).

Apache Spark: A unified engine for big data processing. *Communications of the ACM, 59*(11), 56–65. https://doi.org/10.1145/2934664

39. Zhang, L., Aggarwal, C., & Qi, G.-J. (2017). Stock price prediction via discovering multi-frequency trading patterns. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 2141–2149). ACM. https://doi.org/10.1145/3097983.3098117