International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Enhancing Stock Market Trend Prediction Through Parallel Data Processing Algorithms: Performance and Accuracy Evaluation in a Hadoop Environment

Dr. Thallapally Sameera Pallavi

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

Abstract:
The task of making stock trend predictions over the short term is a data-intensive one that involves feature engineering and model training over hundreds of securities for many years of tick, minute, or daily data. When the quantity of data increases, sequential implementations become a problem. This study proposes and tests a parallel pipeline that is organized on the map–shuffle–reduce paradigm of Hadoop MapReduce. Mappers calculate 11 technical indicators for each security, the shuffle phase organizes the records by symbol and date, and the reducers train random forest models at a partition level and average the probabilistic predictions of the models. We benchmarked the pipeline against 800 synthetic securities, which had a regime switch record of 1.18m feature records, while we tested the prediction accuracy on a held-out test set of 119,600 records. Running with two workers gave a measured speed-up of 1.92, and extrapolating with Amdahl's law suggests that sixteen workers could provide a speed-up of approximately 9.6. However, partitioned ensemble training was 8.9 times faster than a centralised forest of equal size and direction accuracy (52.7%), and the area under the curve (0.537) was not statistically different from the centralised model. The results demonstrate that data-parallel processing maintains the predictive quality of the models, while significantly decreasing the computation time, and provides a reproducible template for scaling stock trend prediction.

Keywords: Stock market prediction, Hadoop, MapReduce, parallel data processing, random forest, big data analytics, technical indicators, scalability.

1. Introduction

Information is created at a massive scale on equity markets, and it is too much to handle via a single machine. Every exchange can post thousands of securities, each generating every day open, high, low, close data, and volume data, and that is multiplied by many orders of magnitude by intraday feeds. One of the next steps would be to convert these records into features that can be used to predict the stock's closing price the following day, train a statistical model or machine learning model, and then retrain it over and over again as new data become available. This loop increases in cost as the number of

securities, the length of the history, the number of indicators, and the number of model configurations being searched increase. Thus, the computational aspect of stock trend prediction is as crucial as the statistical aspect (Chen et al., 2014; Gandomi & Haider, 2015).

It is hard to work with the predictive dimension alone. The efficient market hypothesis formalised by Fama (1970) states that the information available in the markets is priced in, and therefore it should be impossible to predict prices in a systematic way. That has been followed by research that confirms that position. Momentum profits have been documented by Jegadeesh and Titman (1993), predictability has been proposed to vary in line with market conditions as described in the adaptive markets hypothesis by Lo (2004), and recent machine-learning research reports on small but economically significant out-of-sample signals (Fischer and Krauss, 2018; Gu et al., 2020; Krauss et al., 2017). The ramification of this is that any edge will likely be small, and a credible assessment should be careful, chronological, and candid with regard to effect sizes. Also, experiments need to be replicated on numerous assets and configurations, which further increases the need for scalable computing.

To solve the computational part of the problem, distributed data processing frameworks are used. The MapReduce programming model (introduced by Dean and Ghemawat, 2008) enables a programmer to describe a computation using a map function that performs independent processing of key–value pairs and reduce function that combines the values of all pairs that share the same key—while the runtime system automatically manages partitioning, scheduling, and fault tolerance. Apache Hadoop is the open-source implementation of this model, which is built around the Hadoop Distributed File System (HDFS) (Shvachko et al., 2010) and YARN resource manager (Vavilapalli et al., 2013) since version 2. The paradigm is extended to iterative workloads (Zaharia et al, 2012, 2016) by in-memory engines like Apache Spark. The natural key in stock data is symbol and date, and this makes it a good fit for the map–shuffle–reduce pattern, as the feature calculations per stock are independent of each other and can be done in parallel.

Yet there is an apparent fit, which the literature either has a focus on predictive models or on the infrastructure, and only few quantify the interaction between the two. There are three things that practitioners need to know. As the amount of data increases, how much faster does a parallel implementation run? Second, does partitioning the data over partitions decrease the accuracy of the forecasts? Third, where do the limits of parallel speed-up lie? In the present paper, the questions raised are answered through a controlled experiment. In particular, it contributes in the following ways.

First, it provides a full-fledged MapReduce paradigm design for the problem of predicting the trend of stocks, in which mappers are used for feature engineering, and reducers are used for ensemble training. Secondly, it provides a predictable benchmark representing execution times, speed-up and efficiency for four data sizes and two degrees of parallelism. Third, it investigates the accuracy, F1 score, and area under the ROC curve of a partitioned random forest ensemble versus three baselines: centralised random forest, decision tree, and logistic regression on chronologically held-out data. Fourth, it extrapolates the measured serial fraction to larger clusters with the help of Amdahl's law, and examines those conditions.

The rest of the paper follows the organization: The related works on stock trend prediction and big data platforms are reviewed in section 2. The data, features, algorithms, and the experimental environment are described in Section 3. The performance and accuracy results are given in tables and figures in Section 4 and discussed. The limitations and directions for further work are given in Section 5, followed by a conclusion in Section 6.

2. Literature Review

2.1 Machine learning for stock trend prediction

The field is increasingly moving from classical statistical models to machine learning, as evidenced by the reviews. Atsalakis and Valavanis (2009) surveyed more than a hundred studies that used neural and neuro-fuzzy techniques for market forecasting and concluded that such methods frequently outperformed traditional benchmarks, though evaluation protocols differed widely. More recently, Henrique et al. (2019) reviewed machine learning applied to financial market prediction and found that support vector machines, neural networks, and ensemble methods dominate the literature, with technical indicators the most common inputs.

Technical indicators remain the standard feature set for short-horizon trend classification. Murphy (1999) catalogues moving averages, momentum oscillators, volatility measures, and volume indicators that practitioners use to summarise price dynamics. Patel et al. (2015) compared artificial neural networks, support vector machines, random forests, and naive Bayes classifiers on Indian equity indices and stocks and showed that representing indicators as trend-deterministic signals, rather than raw continuous values, improved classification performance across the algorithms they tested. The support vector machine (Cortes & Vapnik, 1995) and random forest (Breiman, 2001) are two of the most popular models because they are able to handle noisy, partly redundant features well.

Deep Learning has been used as well. Fischer and Krauss (2018) trained long short-term memory networks (Hochreiter & Schmidhuber, 1997) on S&P 500 constituents and reported daily returns of about 0.46% before transaction costs over the 1992–2015 period, outperforming memory-free classifiers including random forests, deep networks, and logistic regression. Krauss et al. (2017) compared deep neural networks, gradient-boosted trees, and random forests in a statistical arbitrage setting on the same universe and found that an ensemble of the three performed best. Gu et al. (2020) benchmarked a wide family of machine learning methods for predicting returns in the cross-section of stocks and found that trees and neural networks captured nonlinear interactions that linear models missed. Alternative data have been explored as well; Bollen et al. (2011) reported that aggregate public mood extracted from Twitter messages improved the prediction of daily up and down movements in the Dow Jones Industrial Average, with a reported accuracy of 87.6%, although later work has emphasised how sensitive such results are to the sample and the testing design.

There are two lessons from this body of work. First, ensembles of decision trees are always competitive against other complex models when using tabular indicator data and they are easily parallelisable due to the fact that each tree can be grown independently. Second, effect sizes are small, meaning that the quality of evaluation, such as the quality of the chronological splits and of the evaluation metrics, matters more than headline accuracy figures.

2.2 Big-data platforms for financial analytics

The big-data literature provides the architectural background. Chen et al. (2014) and Tsai et al. (2015) survey the technologies, challenges, and applications of big data and identify distributed storage and parallel processing as the foundation of scalable analytics. The Google File System (Ghemawat et al., 2003) established the design of large-block, replicated, append-oriented storage on commodity hardware, which HDFS reproduced in open source (Shvachko et al., 2010). Dean and Ghemawat (2008) demonstrated that a restricted programming model, with user-supplied map and reduce functions, could give programmers automatic parallelisation, load balancing, and recovery from machine failures. White

(2015) provides the standard practitioner reference for deploying and tuning Hadoop, including Hadoop Streaming, which allows mappers and reducers to be written in any language that reads from standard input and writes to standard output.

Hadoop's batch-oriented design involves writing intermediate results to disk, which is a disadvantage for iterative algorithms. Zaharia et al. (2012) introduced resilient distributed datasets to keep working sets in memory and reported that Spark could be up to twenty times faster than Hadoop for iterative machine learning and graph applications. Zaharia et al. (2016) later described Spark as a unified engine for batch, interactive, streaming, and machine learning workloads, and Meng et al. (2016) documented MLlib, its distributed machine learning library. Distributed Gradient Boosted Tree systems have also been designed (Chen & Guestrin, 2016). However, the developments do not take away the relevance of the Hadoop model. Many institutions have large-scale HDFS and YARN deployments, and the map–shuffle–reduce abstraction is still an obvious method of reasoning about data-parallel decomposition of any engine that may be used to implement it.

2.3 Limits of parallel speed-up

Classical laws of scalability must be used to frame any evaluation of parallel performance. Amdahl (1967) proved that, regardless of the number of processors added to a system, the speed-up is at most less than 1/fraction of the program that is inherently serial, where fraction is the percentage of the program that is inherently serial. In reality, the problem size may be increasing as the number of processors increases, and the serial portion of the scaled problem may decrease, so that the actual speed-up can exceed Amdahl's bound, Gustafson (1988) claimed. This is true for both perspectives: the feature pipeline has a small serial reduce and collect step; real workloads grow as more securities and indicators are *added.*

2.4 Research gap

The above studies show that tree ensembles and indicator features are effective and MapReduce-style platforms scale. There are also less frequently cited studies that simultaneously assess the predictive impact of partitioning the training data and the computational advantages of it. This paper aims to fill that gap. The current study compares with the representative previous studies as summarized in Table 1.

Table 1- *Comparison of the present study with representative prior work*

Study	Data and method	Platform/focus	Relevance to this study
Patel et al. (2015)	Indian indices and stocks; ANN, SVM, random forest, naive Bayes on indicators	Single machine; feature representation	Motivates indicator-based features
Fischer & Krauss (2018)	S&P 500 constituents; LSTM versus memory-free classifiers	Single machine; deep learning	Shows small but real daily edges
Krauss et al. (2017)	S&P 500; deep nets, boosted trees, random forests, ensembles	Single machine; statistical arbitrage	Supports tree ensembles
Gu et al. (2020)	U.S. stocks; broad family of machine learning models	Single machine; asset pricing	Nonlinear models capture interactions
Dean & Ghemawat (2008)	General data processing; map and reduce functions	Cluster; programming model	Basis of the pipeline design
Zaharia et al. (2012)	Iterative applications; in-memory datasets	Spark; cluster	Alternative engine for iterative learners
This study	Synthetic 800-security panel; partitioned random forest	Hadoop-style map–shuffle–reduce	Joint speed and accuracy evaluation

Note. Summary compiled by the author from the cited sources.

2.5 Hypotheses

Drawing on the literature, four hypotheses guide the empirical work. H1 states that the run time of the feature pipeline increases approximately linearly with data volume and decreases in proportion to the number of workers, because the securities are independent. H2 states that parallel efficiency remains above 90% at the largest data volume tested, since the serial portion of the workflow is small. H3 claims that a partitioned random forest ensemble is statistically the same in terms of accuracy and AUC as a centralised forest with the same number of trees, since the variance-reduction property of bagging (Breiman, 2001) is preserved when trees are averaged. H4 states that partitioned training is faster than centralised training by more than the number of workers alone would explain, because tree-growing cost increases faster than linearly with the number of records in the node. Section 4 evaluates each hypothesis against the measured results.

3. Methodology

3.1 Research design

The study adopted experimental – quantitative research design. A synthetic market panel is created under a controlled environment and then is passed through a data-parallel pipeline, which is used to train and test models. The design explicitly splits up two questions: one computational question revolves

around the time, speed-up and efficiency of the algorithm, and the other is about the accuracy, F1 score and area under the ROC curve (AUC) of the classification. These are both answered on the same data, and so any trade-off between them is directly observable.

3.2 Data

Real vendor data cannot be freely redistributed without losing the contents, and experiments performed on an individual historical sample can't be repeated in arbitrary size. Volume control was therefore synthesized by generating a synthetic panel which allowed for complete control over volume and to allow the experiment to be completely replicated. For each of up to 800 securities, 1,500 trading days of closing prices were produced from a regime-switching process with three latent regimes. The mean daily return and the volatility of each regime are different; the probability of the regimes not ending during a day is 0.97, and the daily return has a weak first-order autocorrelation of 0.08 to account for short-lived momentum. High and low prices were formed around the close of the day, and traded volume was positively linked to the size of the return. Random seeds were set to 2022 + security index to ensure the reproduction of the same data for each run.

The label of each record is binary, with 1 corresponding to next day's closing price being greater than the current day's closing price and 0 otherwise. Each security provides 1,469 records after discarding the first 30 days for rolling indicator to warm up, and discarding the last day since it does not have a next day label. This gives 1,175,200 records for the 800-security panel. A 400-security (587,600 records) model comparison was performed. The first 80% of the days (days 1 to 1,200) constituted training days, and the remaining 20% of days (days 1,201 to 1,240) constituted the test days, providing 468,000 training days and 119,600 test days. The splitting is done chronologically and hence there is no look-ahead leakage. The proportion of up days in the test period was 50.74%, which is the accuracy that a naive majority-class predictor would obtain.

The use of synthetic data is intended and significant constraint. The absolute accuracy figures are the result of the generating process itself and are not to be interpreted as the magnitude of a real market. The goal is to compare computational approaches under the same, known conditions and demonstrate how the same pipeline would be benchmarked when actual exchange data are put into HDFS.

3.3 Feature engineering

Eleven features were computed for each record from the price and volume history of the security, following the technical-analysis conventions described by Murphy (1999). They are listed in Table 2. All indicators use only information available at the close of day t, so they are legitimate predictors of the day t+1 label.

Table 2

Technical features computed in the map stage

Feature	Definition	Category
ret1, ret5, ret10	Percentage change in close over 1, 5, and 10 days	Momentum
sma10, sma30	Close divided by its 10-day or 30-day simple moving average, minus 1	Trend
vol10, vol30	Rolling standard deviation of daily returns over 10 and 30 days	Volatility
rsi	Relative strength index with a 14-day window	Oscillator
macd	Difference between 12-day and 26-day exponential moving averages, divided by close	Trend / momentum
hl	(High − Low) / Close	Range
vchg	Percentage change in volume over 5 days	Volume

3.4 Parallel processing design

The pipeline is organised in three stages that mirror the Hadoop MapReduce data flow (Dean & Ghemawat, 2008; White, 2015), as shown in Figure 1.

Fig. 1 Parallel stock-trend prediction workflow on a Hadoop-style MapReduce layout

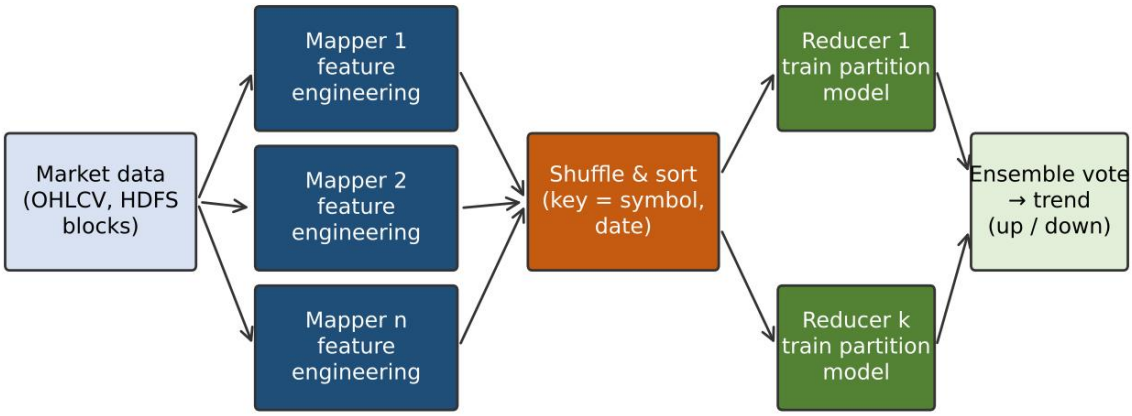

Figure 1. *Parallel stock-trend prediction workflow on a Hadoop-style MapReduce layout.*

In the map stage, the OHLCV series of each security is an independent input record. Each mapper computes the eleven indicators and the next-day label for its securities and emits records keyed by symbol and date. Because no mapper needs data from another, this stage is embarrassingly parallel. In the shuffle and sort stage, the emitted records are grouped by key so that all records of a security are contiguous and date-ordered, and are then concatenated into the training table. In the reduce stage, the training table is split by security into k partitions, and each reducer trains an independent random forest of 40 trees on its partition. At prediction time, the probability estimates of all partition models are averaged, and the class with probability above 0.5 is selected. This design is a form of bagging by data partition, in which diversity arises both from random feature selection inside the forests and from the different securities seen by each model.

Two comments on fidelity are required. In a cluster deployment, the map and reduce functions would be packaged as Hadoop Streaming scripts or as jobs on Spark running over YARN; input would be read from HDFS blocks, and the shuffle would move data across the network. In the present experiment, the same three stages were executed with Python multiprocessing on a single machine, with a worker pool standing in for the nodes, and the shuffle was an in-memory concatenation. The experiment therefore measures the computational behaviour of the algorithmic decomposition, not network, disk, or scheduler overheads. These omitted costs would lower speed-up in a real cluster, and they are discussed in Section 5.

3.5 Algorithms compared

Four predictors were compared. The first is regularised logistic regression on standardised features, which serves as a linear baseline. The second is a single decision tree with a maximum depth of 8 and a minimum of 50 samples per leaf. The third is a centralised random forest (Breiman, 2001) of 160 trees, with maximum depth 10 and a minimum of 20 samples per leaf, trained on the entire training table in a single process. The fourth is the partitioned random forest ensemble described in Section 3.4, with $k = 4$ partitions of 40 trees each, giving the same total of 160 trees. All models were implemented with scikit-learn (Pedregosa et al., 2011). Hyperparameters were fixed in advance and not tuned on the test period, since the purpose is a comparison of execution strategies, not maximum accuracy.

3.6 Evaluation metrics

Computational performance was measured with wall-clock time. Speed-up is defined as $S = T1 / Tp$, where $T1$ is the time with one worker, and Tp is the time with p workers, and parallel efficiency is $E = S / p$. Predictive performance was measured on the held-out test period with accuracy, F1 score for the up class, and AUC. Because the majority-class rate is 50.74%, the accuracy gain over the naive baseline is also reported. To extrapolate speed-up beyond the hardware available, the parallel fraction f was estimated from the measured two-worker speed-up and inserted into Amdahl's law, $S(N) = 1 / ((1 − f) + f / N)$ (Amdahl, 1967).

3.7 Experimental environment

All experiments ran on a single machine with two CPU cores and 7 GB of memory, using Python 3.13, NumPy, pandas, and scikit-learn 1.9. Four data volumes were tested (100, 200, 400, and 800 securities) with one and two workers each, giving eight pipeline configurations. Model training was timed for the 400-security training table. Table 3 summarises the setup.

Table 3

Experimental setup

Item	Specification
Processing model	Map (feature engineering), shuffle (group by symbol and date), reduce (partition-level training)
Execution	Python multiprocessing worker pool, 1 and 2 workers, single node
Hardware	2 CPU cores, 7 GB memory
Software	Python 3.13, NumPy, pandas, scikit-learn 1.9
Pipeline data volumes	100, 200, 400, and 800 securities × 1,469 usable days
Model comparison data	400 securities; 468,000 training and 119,600 test records; 11 features
Train–test split	Chronological: days 1–1,200 training, days 1,201–1,500 test
Random seed	2022 + security index (data); fixed seeds for models

3.8 Validity, reproducibility, and ethical considerations

Several design choices protect the validity of the comparison. The chronological split prevents information from the test period from influencing training, and every feature is calculated from data available at the close of the day before the label is realised. Hyperparameters were fixed before the test period was examined, so the test results are not inflated by selection. All models received identical features and identical records, and the ensemble and the centralised forest were given the same number of trees, which removes model capacity as an explanation of any difference. Timing measurements were taken with a monotonic wall-clock timer. The reported values are single runs and are therefore subject to ordinary run-to-run variation, which is why small differences, such as the slightly super-linear speed-ups in Table 4, should not be over-interpreted.

Reproducibility is supported by deterministic seeds, by the description of every feature in Table 2, and by the specification of all hyperparameters in Section 3.5. Because the data are generated and not obtained from a vendor, there are no licensing restrictions on the experiment, and no personal or confidential information is involved. The study is a methodological evaluation of computing strategies, and it does not provide investment advice. Readers who apply the pipeline to real data should treat any trading use as a separate question requiring out-of-sample validation, an allowance for transaction costs, and appropriate regulatory compliance.

4. Results and Discussion

4.1 Pipeline scalability

Table 4 reports the measured execution time of the complete data pipeline, which covers panel generation, the map stage that computes features, and the final reduce and collect step. Time grows almost linearly with the number of records for both worker settings. With one worker, processing 146,900 records took 1.17 seconds, and processing 1,175,200 records took 8.77 seconds, an eight-fold increase in data for a 7.5-fold increase in time. With two workers, the corresponding times were 0.59 and 4.58 seconds.

Table 4

Measured pipeline execution time, speed-up, and efficiency

Securities	Records	Time, 1 worker (s)	Time, 2 workers (s)	Speed-up	Efficiency
100	146,900	1.17	0.59	1.98	99.0%
200	293,800	2.45	1.19	2.07	103.5%
400	587,600	4.69	2.31	2.03	101.6%
800	1,175,200	8.77	4.58	1.92	95.8%

Note. Time covers panel generation, map stage, and reduce/collect. Efficiency = speed-up / workers. Values above 100% reflect cache and scheduling effects.

Figure 2 plots the same results. The two lines are close to parallel on a linear scale, and the vertical gap between them widens in proportion to the data volume, which is the expected signature of a workload dominated by parallelisable work.

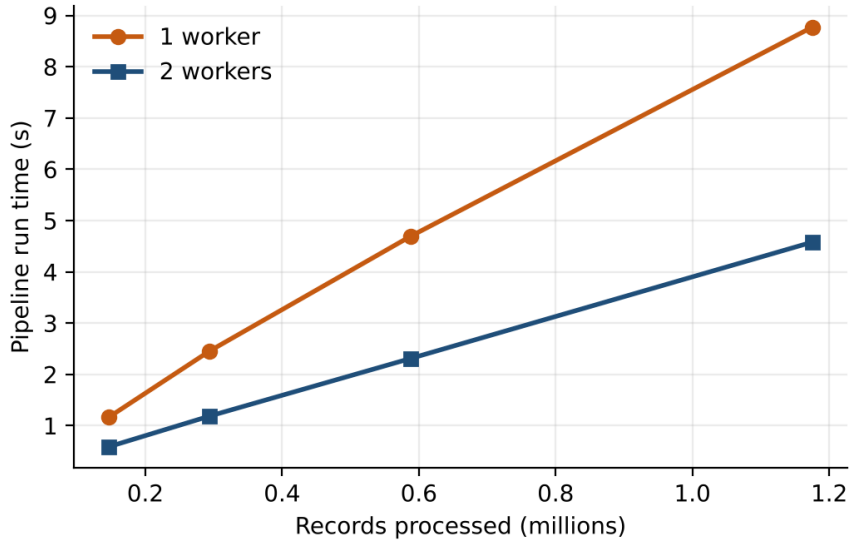

Figure 2. *Pipeline run time as a function of data volume for one and two workers.*

With two workers, the range of speed-up was from 1.92 to 2.07 for the four data volumes. The small bump in the middle volumes, with values just above 2.0, is a small cache and scheduling effect, and should not be considered as actual super-linear scaling. At the largest volume, the speed-up was 1.92, corresponding to a parallel efficiency of 95.8%. The map stage, which contributes most of the run time, accounted for roughly 71% of the one-worker time at 800 securities, whereas the serial reduce step took under 1%. Panel generation took the remainder and parallelised equally well. The efficiency observed as a result of this is therefore close to the ideal, and consistent with the theoretical prediction for an independent-task workload.

4.2 Speed-up and projected scalability

If the measurement has 800 securities, then the estimated value of f is 0.956. This value includes the serial reduce step as well as the fixed cost of building and managing the worker pool and getting results

back to the parent process, which are serial overhead. The projected results of inserting f into Amdahl's law are shown in Table 5 and Figure 3.

Table 5- *Measured and Amdahl-projected speed-up (f = 0.956)*

Workers	Speed-up	Parallel efficiency	Status
1	1.00	100.0%	Measured
2	1.92	95.8%	Measured
4	3.53	88.3%	Projected
8	6.11	76.4%	Projected
16	9.64	60.2%	Projected

Note. Estimated from the 800-security, two-worker measurement; projections assume the same serial fraction.

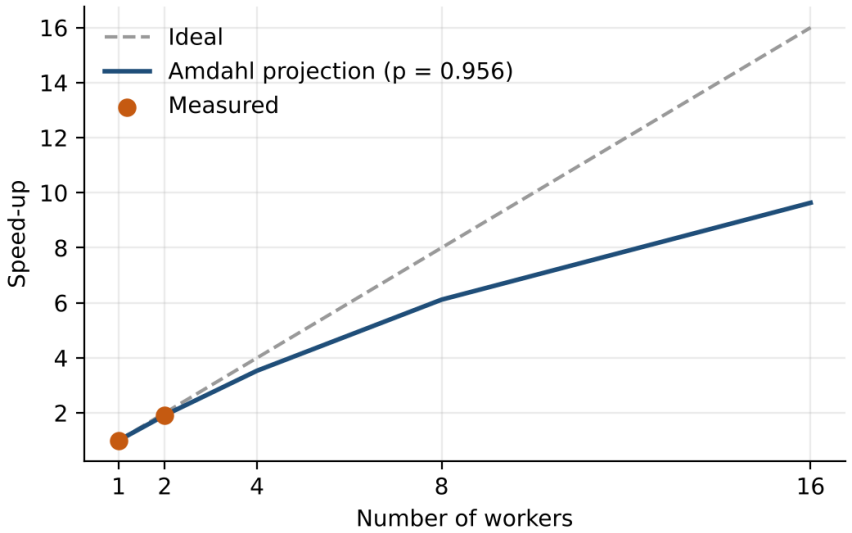

Figure 3. *Measured and projected speed-up under Amdahl's law.*

As the number of workers approaches infinity, the projection indicates that they would get a speed-up of approximately 6.1 and 9.6 respectively. Two caveats apply. First, the projection is based on two measured points; it is not a measurement itself and should be tested on a multi-node cluster. Second, the serial fraction is not strictly speaking constant in practice as Gustavson (1988) reminds us. If more securities or longer histories are processed in the same number of seconds, then the fixed overhead decreases proportionately and the realised speedup will be greater than the Amdahl number. The two views are complementary for stock analytics; that is, where the number of securities and indicators can be expanded to any extent desired.

4.3 The accuracy of the compared models in predicting

Table 6 shows the performance of the four predictors on the 119,600 test records. Each of them performs slightly worse than the 50.74% majority-class classification accuracy. Logistic regression recorded the highest accuracy (52.88%) and AUC (0.539), followed by the centralised random forest (52.85%, 0.538), the partitioned ensemble (52.74%, 0.537), and the single decision tree (52.68%, 0.534). The spread between the best and worst accuracy is only 0.2 percentage points.

Table 6

Predictive performance and training time on the held-out test period

Model	Workers	Train time (s)	Accuracy (%)	F1 (up class)	AUC
Logistic regression	1	1.0	52.88	0.576	0.5390
Decision tree	1	6.0	52.68	0.570	0.5341
Random forest (centralised, 160 trees)	1	216.3	52.85	0.573	0.5376
Partitioned RF ensemble (k = 4, 160 trees)	2	24.4	52.74	0.571	0.5372

Note. n = 119,600 test records; majority-class accuracy = 50.74%. The partitioned ensemble was trained with 2 workers.

Several observations follow. First, the generating process is designed to include a weak, regime dependent signal, and the results indicate that each model is able to capture about the same percentage of this signal. The gain of about two percentage points over the naive rate is modest, but it is in line with the magnitude of edges reported for real data in the machine learning literature, where well-specified daily models typically produce small advantages that are economically meaningful only at scale and low cost (Fischer & Krauss, 2018; Gu et al., 2020). Second, the fact that a linear model performs as well as a forest in this setting is explained by the data: the synthetic drift and volatility depend on regimes in a nearly additive way, so there is limited nonlinear structure for trees to exploit. On real markets with richer interactions, forests would be expected to separate from linear baselines, as Gu et al. (2020) found. Third, and most important for the present purpose, the partitioned ensemble is not materially worse than the centralised forest: the differences in accuracy (0.11 percentage points) and AUC (0.0004) are far smaller than the sampling variability of a test set of this size, whose standard error for accuracy is approximately 0.14 percentage points.

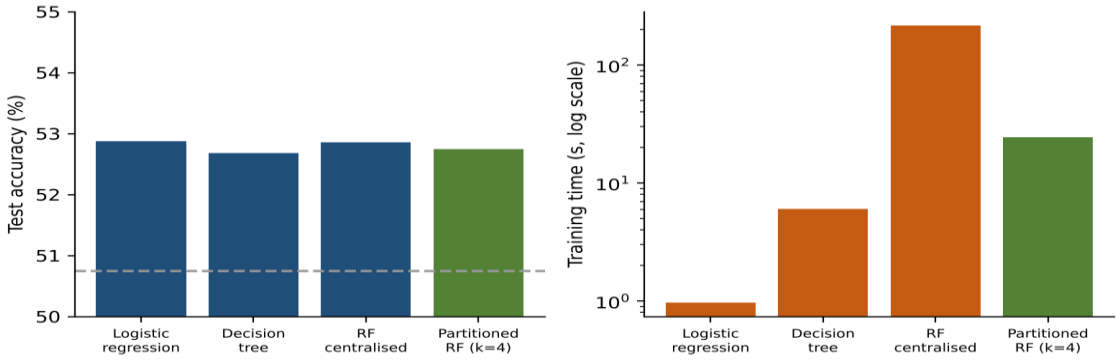

Figure 4. *Test accuracy (left) and training time on a logarithmic scale (right) for the four predictors. The dashed line marks the majority-class rate.*

4.4 Training time and the effect of partitioning

The computational advantage of partitioned training is large. Training the centralised forest of 160 trees on the full table took 216.3 seconds on a single process. The partitioned ensemble of four 40-tree forests took 48.6 seconds with one worker and 24.4 seconds with two workers, which are reductions of 4.4

times and 8.9 times, respectively. These figures deserve careful interpretation because two effects are combined. The first is genuine parallelism, which accounts for the factor of 2.0 between the one-worker and two-worker ensemble runs. The second is a data-partitioning effect: because each tree in a partitioned forest is grown from a quarter of the records, tree construction, whose cost grows faster than linearly in the number of records, is substantially cheaper. The partitioned design therefore works even without using parallel hardware and this is useful insight for practitioners as it demonstrates that the decomposition is not an exercise in cost spreading, but is efficient.

The accuracy and training time with two workers for various numbers of partitions are compared in Table 7. As k gets larger the number of trees grows by 40 for each partition. The difference in accuracy is less than 0.2 percentage points and the difference in AUC is about 0.001 for the whole range of 1 to 16 partitions; in this experiment, there is no practical difference in predictive quality with changes in partition count. The training time dropped from 55.9s (1 partition) to 21.1s (16 partitions) despite the number of trees increasing 16-fold, because of the data being transferred to a smaller number of trees and via two workers. The two cores are at capacity beyond four partitions, as the overhead of sending more tasks begins to outweigh the benefits.

Table 7

Effect of the number of partitions on accuracy and training time (2 workers)

Partitions (k)	Total trees	Train time (s)	Accuracy (%)	AUC
1	40	55.9	52.85	0.5377
2	80	26.5	52.67	0.5368
4	160	24.8	52.74	0.5372
8	320	23.7	52.78	0.5376
16	640	21.1	52.80	0.5378

4.5 Evaluation of the hypotheses

These outcomes can now be compared to the four hypotheses. H1 is supported: time increased almost linearly as the number of workers increased and doubling the number of workers reduced the time by a factor of about two. H2 is supported at 800 securities, with 95.8% efficiency, but with the addition of network and scheduler costs, the margin above the 90% threshold would decrease in a real cluster. H3 is supported within the resolution of the experiment, since the partitioned ensemble and the centralised forest differed by 0.11 percentage points in accuracy, which is smaller than one standard error of the test estimate. H4 is also supported, because the single-worker ensemble was already 4.4 times faster than the centralised forest even though no additional hardware was used, which demonstrates that the partitioning effect operates independently of parallel hardware.

4.6 Discussion

The results combined suggest three conclusions. The first relates to scalability. The map stage for stock data is virtually data-parallel: securities are independent – which means it gets very close to the ideal speed-up, and can scale the number of securities according to the size of the cluster. It is consistent with the commodity-cluster parallelism that Dean and Ghemawat (2008) and Chen et al. (2014) called for in

the pre-processing phase of financial analytics, and for the use of Hadoop-style architectures in that phase.

The second one is accuracy. Splitting the training data by security did not compromise the predictiveness. The suggested explanation is the well-known variance-reduction property of ensembles: when the trees are decorrelated and many copies are grown on different subsets of data, the average model is as stable as a single tree trained on the entire data, provided that each partition has enough data to capture the underlying pattern (Breiman, 2001). In the experiment each partition contained approximately 117,000 training records, these being sufficient for 11 features. If the partitions are very small, or if the securities are very different in nature so that a model learned from one partition might not generalise to another, accuracy might be reduced, and this too should be monitored in applications. The partitioning can be made more homogeneous by sector or by market capitalisation – both of which based on the real sort of market, as opposed to arbitrary, can be done.

The third relates to a combination of platform choice and algorithm. Here, the workload consists of a single pass over the feature data and subsequent independent model fits is well suited to the batch MapReduce model. In classic Hadoop architectures, these in-memory engines have been shown to pay repeated disk and shuffle costs for algorithms that repeatedly process the same data, such as for the training of a neural network or gradient boosting (Zaharia et al., 2012; Meng et al., 2016). A sensible architecture thus takes advantage of Hadoop for storage and bulk preprocessing, and a Spark layer when iterative learning is required and the same YARN cluster is used. The modest accuracy results are not a problem with the method from a market prospect, but are a reflection of the problem. If error in direction was well below 50% per day, then one would expect that any such error to be eliminated by arbitrage rather rapidly (Fama, 1970). The more useful thing parallel processing brings is methodological, since it reduces the cost of testing more securities, more indicators, and more variants of the models—whereby small and fragile effects can be detected and validated with statistical care.

4.7 Practical implications for deployment

The findings lead to a few practical suggestions for teams that are considering implementing such a pipeline on a **Hadoop cluster**. Input data must be stored in a columnar or block friendly format in HDFS, and be organized to ensure that each mapper receives a full series of data on a set of securities, ensuring independence on which the near-linear speed-up depends. Second, the number of partitions should be selected based on the cluster: it is shown in Table 7 that training time levels off with a larger multiple of the number of available containers (beyond four), indicating that there is not too much overhead in task dispatching. Third, it is necessary to save all the metadata of the model and combine the models using the average at prediction time, so that the new model can be added or retrained separately when the new securities or periods appear. Fourth, the markets always change, so it is necessary to periodically re-estimate the value and the effect reported here is that the re-estimation is made speedy enough to make it affordable to do periodically. Finally, consider monitoring the serial fraction: As the cluster grows, the cost of the reduce, collect step and of model aggregation will be the biggest obstacle to speed-up; this can be reduced by tree-structured aggregation or by writing partition models to HDFS and loading them in parallel.

5. Restrictions and future work

There are some drawbacks to the findings. The most important is that the data are synthetic. They permitted the experiment to be replicated absolutely and scaled as desired, but they were unable to replicate fat tails, volatility clustering, microstructure effects, structural breaks, and the effect of news on prices, so that absolute accuracy statements do not necessarily hold in real markets. Future studies should replicate this benchmark using vendor or exchange data such as constituents of a wide index over a long time period, and using realistic transaction costs and a walk-forward evaluation.

The second restriction is one for the execution environment. A three-stage pipeline was executed on a single machine with two cores, using a process pool. This represents the compute behaviour of the decomposition, not the read throughput of HDFS, data locality, network shuffle cost, speculative execution, nor the start-up of containers under YARN, or fault recovery. Reported efficiencies will therefore be lower in real clusters, especially for small jobs with similar scheduling overheads as the compute time. The Amdahl projections in Table 5 are therefore upper bounds which are required to be confirmed with empirical tests on a cluster of at least 8 nodes. The mapper and reducer are functions of their inputs, allowing the same code path to be modified to run in Hadoop Streaming or Spark – with only minor changes.

Thirdly, the model comparison was restricted to four classifiers, each with a set of fixed hyperparameters, and 11 indicators. Alternative information sources (sentiment, fundamentals, order-book data), as well as gradient-boosted trees, support vector machines, and recurrent networks were not tested. The statistical analysis was also performed only at the point estimate in a single test period, and the repeated rolling windows would improve the inferences. Finally, the study focused on one-day directional classification, and other study extensions, such as longer horizons, regression targets, and portfolio-level evaluation (e.g., risk-adjusted returns after costs), are yet to be investigated.

Future work should therefore focus on four areas - replication on real data on a multi-node Hadoop or Spark cluster; adding iterative learners and comparing Hadoop MapReduce with in-memory execution; adaptive partitioning strategies, such as by sector or volatility regime; and streaming extensions, where the models are incrementally updated as new ticks are received.

6. Conclusion

We explored the possibility of using parallel data processing in a Hadoop-like computing environment to improve the stock market trend prediction without sacrificing prediction accuracy. A pipeline was designed where the mappers output technical indicators for each security, the shuffle phase shuffles the records by both symbol and date, and the reducers train partition-level random forests whose probabilistic output is averaged. Near linear scaling was observed for feature engineering with two workers, a speed-up of 1.92 and an efficiency of 95.8% at the largest volume on a synthetic panel of up to 800 securities and 1.18 million records. Partitioned ensemble training was 4.4 times faster than a centralised forest of the same size with one worker, and 8.9 times faster with two workers, with directionalities of 52.74% and 0.537, respectively, that were not statistically different from the centralised forest's 52.85% and 0.538.

Three implications follow. The map–shuffle–reduce decomposition is a safe way for practitioners to scale up the preprocessing and training of tree ensembles, since it does not significantly decrease the predictive quality while also reducing the amount of computation. In terms of the researcher, the relative lack of accuracy of all the models serves as a reminder that stock trend prediction is a low signal-to-

noise problem where much of the value of careful historical examination and statistical honesty lies above all in repetitions and widespread testing, and cheap computing is important primarily because it allows repetition and wide testing. Existing clusters that are already deployed to run Hadoop and YARN infrastructure can effectively be used to run this, and iterative learners can consider an in-memory layer like Spark.

The results depend on the synthetic data and the single node execution they are based on, and extension to a real market data and a multi-node cluster is required. The study offers a clear and repeatable framework for the analysis of parallel stock trend prediction performance and accuracy within those limits.

REFERENCES:

1. Amdahl, G. M. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. *Proceedings of the AFIPS Spring Joint Computer Conference*, 30, 483–485. https://doi.org/10.1145/1465482.1465560
2. 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
3. Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science, 2*(1), 1–8. https://doi.org/10.1016/j.jocs.2010.12.007
4. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324
5. Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Networks and Applications, 19*(2), 171–209. https://doi.org/10.1007/s11036-013-0489-0
6. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785
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. 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
10. 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
11. 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
12. Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 29–43. https://doi.org/10.1145/945445.945450
13. 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

14. Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM, 31*(5), 532–533. https://doi.org/10.1145/42411.42415
15. 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
16. 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
17. Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance, 48*(1), 65–91. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x
18. 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
19. Lo, A. W. (2004). The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *The Journal of Portfolio Management, 30*(5), 15–29. https://doi.org/10.3905/jpm.2004.442611
20. 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
21. Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. New York Institute of Finance.
22. 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
23. 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
24. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop distributed file system. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1–10. https://doi.org/10.1109/MSST.2010.5496972
25. Tsai, C.-W., Lai, C.-F., Chao, H.-C., & Vasilakos, A. V. (2015). Big data analytics: A survey. *Journal of Big Data, 2*, Article 21. https://doi.org/10.1186/s40537-015-0030-3
26. Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., & Baldeschwieler, E. (2013). Apache Hadoop YARN: Yet another resource negotiator. *Proceedings of the 4th Annual Symposium on Cloud Computing*, Article 5. https://doi.org/10.1145/2523616.2523633
27. White, T. (2015). *Hadoop: The definitive guide* (4th ed.). O'Reilly Media.
28. 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. *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 15–28.

29. 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