OXFORD JOURNAL
OF STUDENT
SCHOLARSHIP

Comparative Analysis and Ensemble Optimization of Machine Learning Models for Next-Day Stock Price Prediction

Yiwei Max Ye
lhdc146@gmail.com

ABSTRACT

Predicting stock market movements remains a complex challenge due to the market's inherent volatility, noise, and sensitivity to both fundamental information and investor behavior. This study investigates the application of multiple machine learning models for next-day stock price prediction by utilizing five years of historical daily opening price data obtained from Yahoo Finance. A rolling three-day input window was used to forecast the fourth day's price. Five models– linear regression, decision tree regression, random forest regression, multi-layer perceptron (MLP), and k-nearest neighbors (KNN) –were trained and tested using a training-testing split, with early stopping implemented to reduce overfitting and improve generalization. Model performance was assessed based on mean squared error (MSE). Ultimately, the simulation demonstrated consistent profitability.

The results show that while particular models exhibit varying levels of accuracy, combining models can improve robustness and reduce bias towards certain companies. However, there are still limitations, particularly due to the reliance on historical price data and the inability to fully capture external shocks, current events, and structural changes in financial markets. These findings highlight both the potential and the constraints of pure quantitative machine learning in the stock prediction field, suggesting that future approaches should incorporate more adaptive and qualitative aware methodologies.

INTRODUCTION

Predicting stock market prices has been a challenge, but also a very tempting problem in the field of finance. However, the market is always noisy–constantly changing not only due to technical and informational factors, but also due to temporal background, fear and greed, emotional factors, and many other influences. In recent years, machine learning has become a more popular approach to such challenges because of its pattern recognition, various network information associations, and large-scale data analysis capabilities. Though a strong dataset and logical output do not necessarily mirror real-world market performance, a unitary model, biased weighting, overfitting, and lack of attention could all result in an evaluation that contains hidden bias. This essay investigates how a machine learning model

predicted a single company's market movement, with comparisons among different training algorithms, hyperparameter tuning, and fine-tuning.

The stock market was first established in Amsterdam, the Netherlands, in 1602 by the Dutch East India Company as the Amsterdam Exchange. In 1602, exactly at the time of the Age of Discovery, colonization had granted Western nations huge amounts of capital. Therefore, the invention of the stock market was bound to happen. This invention marked the transformation of the Western world from mercantilism to capitalism. Capitalism, a market-based economy, heavily relies on the term "market," and the stock market, though not fully representing the market, is a figurative existence of it. Thus, the stock market often represents a capitalist nation's economic ebb and flow.

A privately owned company could go public on the stock market; after appearing on the market, the company is able to be traded through shares, and each shareholder possesses a part of the company. In the market, stock traders could also short stocks, predicting that the market will go down. Therefore, predicting whether a stock will go up or down is tempting. However, the value of a share changes for various reasons: a company's new product, changes in the board of directors, decision-making, current global events, cash flow, profitability, and the country's bank interest rate, among others.

Indeed, even with all these structures, the stock market is difficult to predict because the price responds to both information and behavior. In theory, the market should immediately "price in" any public news due to the invisible hand theory. In reality, prices can move irrationally in the short term because of investor emotion, herd effect, and sudden shifts in confidence, especially under major announcements such as earnings and unexpected global events such as tariffs. This creates short-term uncertainty in which the market demonstrates abrupt changes. Ultimately, the market exhibits fluctuating characteristics, even if the long-term trend reflects the company's fundamental factors. Therefore, finding the balance between long-term factors and short-term emotion is the key challenge in stock prediction and a reason data-driven models have become effective.

Artificial intelligence, often referred to as AI, is a branch of computer science that creates systems capable of reacting to tasks like humans: learning, problem-solving, perception, and decision-making. Instead of programming the next step, AI often learns patterns from previous datasets and improves itself over time based on its performance. Nowadays, people interact with generative AI the most, but the key basis of AI is actually machine learning. Machine learning is the discipline in which an algorithm or model learns from a dataset and predicts patterns. In our project, multiple machine learning models are applied to find the next day's stock market price.

In this essay, Zou et al. (2023) conclude that the current stock market prediction models, up to 2023, are mostly traditional machine learning models but lack the application of deep learning. In their research, there are three metrics used to measure model performance: error-based, the difference between predicted and actual price; accuracy-based; and return-based. Ultimately, deep learning has become central in stock prediction research. The future of machine learning for stock market prediction will evolve around deep learning.

MATERIALS AND METHODS

In this project, we use the previous five years of daily stock market prices as our dataset, which was imported from Yahoo Finance. These five years of stock market prices are the opening prices of each day. During our training, we create a three-day window across 1,260 days. Each three-day dataset is used to predict the fourth day, and so on over the 1,260 days.

In the training process, we split the dataset into the training set and testing set, with 33% for the testing set and 67% for the training set. The key is that the model trains itself on the training set and is eventually tested on the testing set to calculate its mean squared error. This is like a student learning a topic: they first use a worksheet and practice problems to retain the knowledge, and later they take a test to assess their knowledge of the unit. Because stock prices form a time series, the split was therefore performed chronologically rather than by random shuffling: the earlier portion of the data was used to train the models, and the later portion was used to test the models. This preserves temporal order and reduces the risk of information leakage from future observations into model training. We feed the following five different machine learning models with the same training and testing sets. From that, we calculate their mean squared errors with respect to these training and testing datasets.

Linear regression is the first and most familiar prediction model people normally use. It is a linear prediction model. The output is the weighted average of our input (see Figure 1).

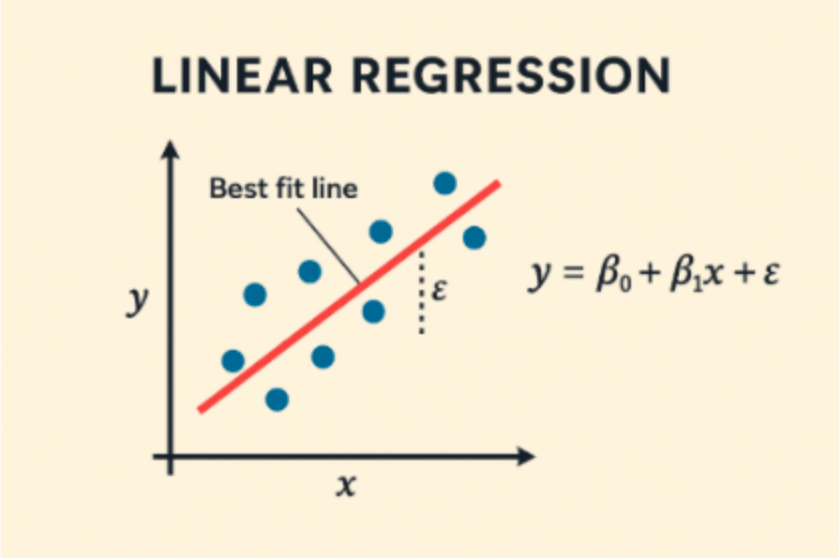

Figure 1. Linear regression diagram showing the best-fit line and residual error (ε)

A decision tree regressor recursively acts as a command-based condition-making system. Through these conditions, the children of the parent node narrow the input down and eventually fit constant predictions in the terminal leaves. Its behavior is governed by hyperparameters, such as its max depth, or how many levels the tree can grow; assume these are the layers of a neural network. Because very deep trees can cause overfitting–a graph that fits the data accurately but predicts it incorrectly–like in real life, if a system has too many conditions, it often cannot run in a realistic scenario even though it works theoretically. We perform hyperparameter tuning, the process of selecting values such as depth that optimize validation performance in our code (see Figure 2).

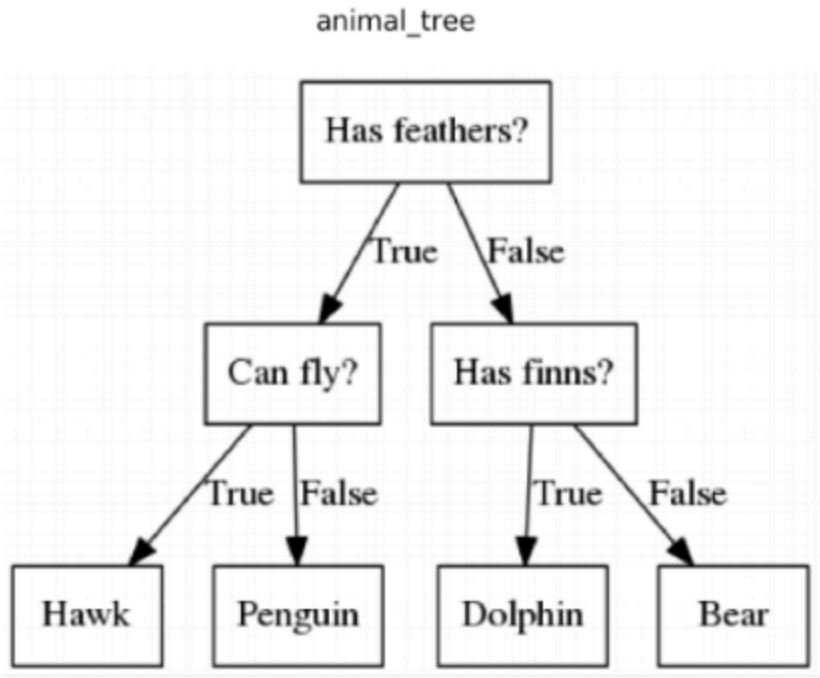

Figure 2. Example of a decision tree structure on animal classification

A random forest regressor sums the predictions of many decision trees that are trained on different samples of the data, with feature randomness. Then, linear regression is applied to all of these random outputs, which serve as the input for the forest. Ultimately, the forest's output is the average of its trees' predictions. Key hyperparameters therefore include the number of trees and the maximum depth of each tree, both of which control the bias of this system (see Figure 3).

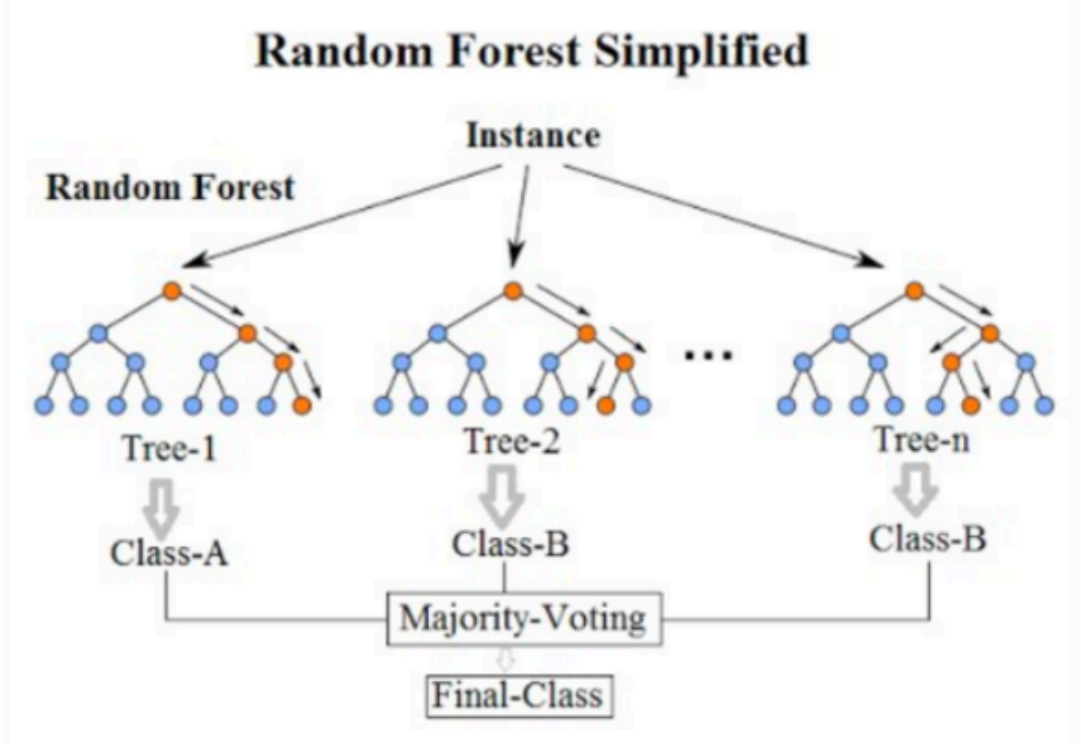

Figure 3. Simplified diagram of a random forest model showing multiple decision trees and majority-voting for a final classification.

MLP (Multi-Layer Perceptron Regression) is the fourth model we introduce in this project. Multi-layer perceptron regression is a set of neural networks consisting of an input layer, hidden layers, and an output layer. Inside these hidden layers, the MLP model maps continuous input data to a continuous numerical output by passing data through nonlinear activation. The more parameters we have, the more complexity there is in these hidden layers (see Figure 4).

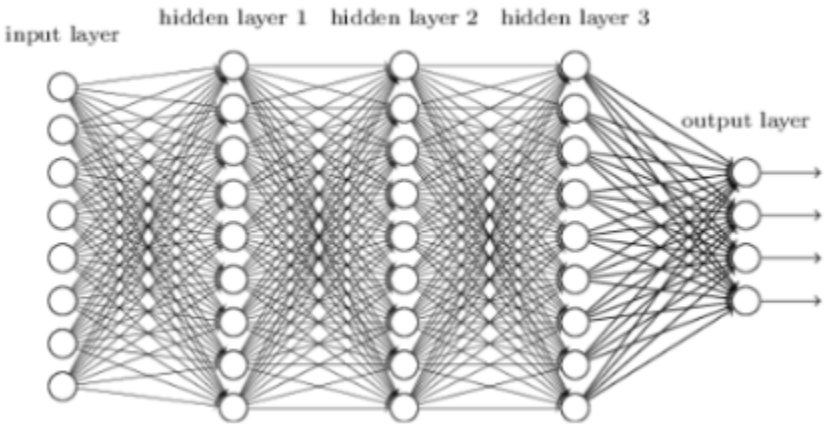

Figure 4. Architecture of a multi-layer perceptron (MLP) showing the input layer, three hidden layers, and output layer.

K-nearest neighbor is the fifth model we use in the project to predict the next -day market price. KNN (K-nearest neighbor) is a machine learning algorithm that takes multiple input datasets and makes a prediction. This prediction is based on a tunable hyperparameter, for example, five data points. The final prediction will be the majority similar result from the dataset. For example, if we have a dog-and-cat identifying model that implements KNN, when it comes to making the prediction of whether something is a cat or a dog, this model will set a zone for distinguishing whether it is a cat or a dog from the dataset. This zone is the hyperparameter we mentioned earlier. In this zone, if the similar results are mostly dog, then the final output equals dog, and vice versa (see Figure 5).

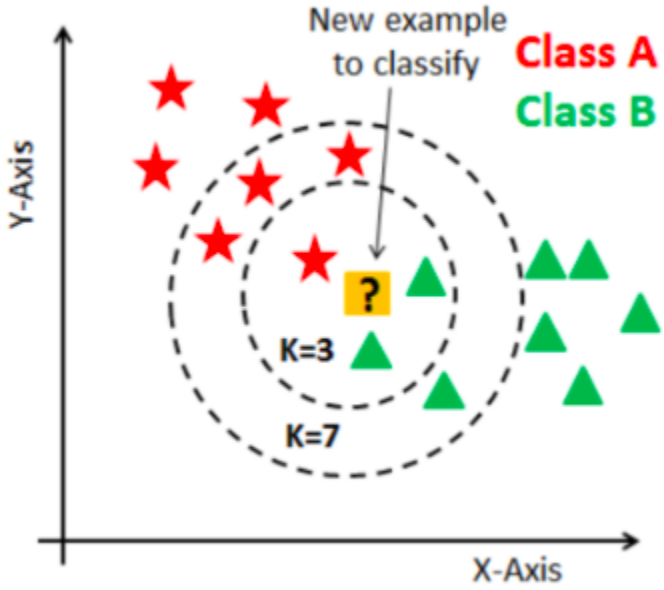

Figure 5. K-nearest neighbor classification diagram illustrating the decision boundaries for two classes.

After simulating, we want to base the weights assigned to the models on the mean squared error calculated from each model in order to find the best system for predicting stock market price. The larger the mean squared error each model has, the less weight it is assigned in the final output.

By assigning weight, we do increase the performance of our model by decreasing the bias across machine learning models. However, the neural network system often struggles when interacting with multiple models' outputs and eventually combining them into one comprehensive answer. Here is where weight comes into play: standard deep learning models are often given averaged static weights distributed based on mean squared error, the error calculated from the difference between the predicted output and the actual data.

In a traditional neural network system, these weights often do not work well. This uniform weight distribution creates inefficiency in passing information from one layer to the next, where the network

cannot dynamically relocate its capacity. The model is therefore locked into adapting a single set of weights that must simultaneously work with diverse input, leading to a result in which no particular type of input gets an optimal solution.

Another critical limitation of average weighting comes from the model's inability to distinguish between fundamentally different input characteristics during inference. When a neural network is trained on highly diverse data, this static weighting processes all input identically. This means the neural network cannot specify its input parameters based on the input type. The weight becomes averaged in all cases, resulting in a system that settles for a low bar but never masters anything.

Lastly, training with traditional weighting will also perform poorly on specific tasks. For example, a model may be good in a specific situation. In this case, we cannot choose to ignore this model in normal cases; we have to distribute a weight to it. It will not be accurate because in specific cases it performs unexpectedly well but gets weighted too lightly, while in normal cases it gets weighted too heavily. This case will always happen because of the weight distribution's inability to turn the model on and off. This means that whenever we use this method, we have to coordinate the entire network, which brings unnecessary training cost and computing power loss.

A way to solve the limitations of average weighting is to employ a mixture expert method. A mixture expert could be understood as a consultant nurse at a hospital front desk: it could briefly diagnose what symptoms the patient has; however, it would not give a prescription, but it would guide the patient to the expert doctor for the problem the patient is having. The mixture expert method adjusts the weight of each following model based on the output from the previous network. Thus, the weight of the final merging system is flexible.

The benefit of using a mixture expert is that it is able to avoid all the disadvantages that weight assigning brings while successfully merging all the model outputs into one more comprehensive output. The mixture expert could also be limited for two reasons: the first set of outputs may be poor, and the combining expert, the consultant, may also be poor. The ways to fix these issues are either removing the outlier bad-performing model or using a more complicated network–instead of hard-coding the combiner as linear, we train another neural network to decide how to combine the experts.

In this study, we implemented a stacking-based ensemble approach related to the mixture expert's framework. First, several base models generated predictions from the same stock-price input windows. Next, those predictions were used as inputs to a second-stage meta-model that learned how to combine them into a final prediction. The meta-models tested in this second stage included linear regression, decision tree regression, random forest regression, K-nearest neighbors, and an MLP regressor.

In addition, we also implement early stopping as a method in the optimization process. Early stopping is a machine learning training strategy. Training a machine learning model normally goes through multiple epochs; an epoch is when the model makes one full pass through the entire training dataset. As the number of epochs increases, the model's performance first increases; however, after a certain point, it tends to show overfitting—the performance increases on the training set, but worsens on unseen validation data. As a result, early stopping is used to put a checkpoint at the point at which the model starts overfitting, making sure the model does not "learn too much" and therefore avoids overfitting.

The key idea behind early stopping is to monitor a model's performance on a validation set before entering the test phase, which uses a separate dataset that the model does not train on. The validation set acts as a practice test for the model to generalize its performance. If performance stops increasing on the validation set—for example, if the performance of the model does not improve by a certain percentage after going through an epoch—then the stopping point is found, and the training stops. This prevents continued parameter updates that mainly benefit the training set, and it also saves computation time by not running extra epochs once progress has essentially reached the convergence point, once the improvement plateaus.

RESULTS

To assess these models' performance, we establish a simulation strategy for a stock market buy-and-sell system focusing on one company's stock; here, we use Tesla. In this system, we use different machine learning models to predict the next day's price over 1,210 days, roughly the number of days the market opens in five years. It starts with a 10,000-dollar budget and owning 0 shares of stock. For each of the 1,210 days, it checks whether the model predicts the next day would be higher or lower than the current price: if the model predicts that the next day's price will be higher than the current day's price, the system will buy two shares, (if sufficient cash is available). If the model predicts that the next day's price will be lower, the system sells two shares (only if at least two shares are currently held). On the last day, it sells all the stock, thereby transforming all the capital from shares to cash. Ultimately, it prints the total amount of money after the simulation. From this simulation, our initial money increased from $10,000 to $32,724; evidently, the model functioned as we intended it to, making an impressive profit in most companies compared to the buy-and-hold strategy and random trading . We also tried a more aggressive simulation strategy by buying and selling five shares of stock per day. Indeed, through simulation, the more aggressive method earns more revenue, from $33,176 to $40,781, proving that the model's accuracy is considerable. Here, Table 1 shows the training MSE across all five models and companies, Table 2 shows the testing MSE, and Table 3 shows the revenue from 2-share strategy.

To provide context for the trading results, we compared the model-driven strategy against benchmark strategies, including buy-and-hold and random trading. These benchmarks help evaluate whether the model-based system outperforms simpler alternatives rather than appearing profitable only because of the underlying stock's long-term upward movement.

Table 1: Train Mean Squared Error (MSE) Across Five Models and Five Companies

Train Mean Squared Error	Linear Regression	Decision Tree	Random Forest	Multi-layer Perception	K-nearest neighbor
Tesla	114.57	733.87	571.54	144.42	62.4
Oracle	13.21	250.44	210.74	16.33	7.45
Chipotle	0.83	11.25	9.71	1.08	0.43
Eli Lilly	178.53	4937.87	3674.05	213.73	96.76
McDonald	8.94	162.05	140.17	11.75	4.52

Table 2 : Test Mean Squared Error (MSE) Across Five Models and Five Companies

Test Mean Squared Error	Linear Regression	Decision Tree	Random Forest	Multi-layer Perception	K-nearest neighbor
Tesla	102.1	1110.08	923.29	131.37	171.47
Oracle	32.3	330.67	273.12	41.89	62.75
Chipotle	0.76	21.41	19.56	1.00	1.36
Eli Lilly	144.89	6030.99	4589.49	165.49	239.33
McDonald	7.14	167.33	147.07	10.55	14.21

Table 3: Simulated Revenue per Company Using a 2-Share Buy/Sell Strategy with $10,000 Starting Capital

Simulated Revenue(Buy and Sell 2 Stock a day)	Input	Output	Revenue
Tesla	10000	32724.24	22724.24
Oracle	10000	20461.00	10461.00
Chipotle	10000	18524.98	8524.98
Eli Lilly	10000	50284.49	40284.49
McDonald	10000	14890.88	4890.88

Table 4: Simulated Revenue per Company Using a 5-Year Buy-and-Hold Strategy with $10,000 Starting Capital

Buy-and-Hold	Input	Output	Revenue
Tesla	10000	21670	11670
Oracle	10000	23122	13122
Chipotle	10000	11290	1290
Eli Lilly	10000	24655	14655
McDonald	10000	11285	1285

Table 5: Simulated Revenue per Company Using a Random-Trading Strategy with $10,000 Starting CapitalDiscussion

Random-Trading	Input	Output	Revenue
Tesla	10000	14596.16	4596.16
Oracle	10000	14548.87	4548.87
Chipotle	10000	6502.32	-3497.68
Eli Lilly	10000	16598.85	6598.85
McDonald	10000	12373.43	2373.43

DISCUSSION

The results suggest that simpler models such as linear regression generalized more reliably than complex models such as KNN, indicating that the dataset's limited variation of input may favor smoother decision boundaries over proximity-based learning. Even with strong simulated performance, an important question remains: can a model trained mainly on past prices generalize to real markets where volatility, news shocks, and investor psychology can abruptly change the "rules" the model learned? If the model is tested across multiple companies and market regimes, would the same strategy still outperform a simple baseline, or would the advantage disappear once conditions shift? In general, the next challenge to solve in this area would be how to efficiently implement a quantifying model for qualitative information for the market.

CONCLUSION

In this project, we researched whether machine learning could be applied to predict next-day stock prices over five years of historical stock market opening prices from Yahoo Finance. We established a three-day input window to predict the fourth day's price. In the training process, we split the dataset into training and testing sets, then trained them through multiple models, including linear regression, decision trees, random forests, an MLP regressor, and K-nearest neighbors. Model performance was evaluated using the mean squared error from each model and a simulation that applied a rule-based strategy to buy or sell stock based on the prediction of the next day's price. We also explored different methods to optimize and combine the models to improve reliability and performance, such as hyperparameter tuning, weight assigning, mixture expert, and early stopping.

REFERENCES

Bishop, C.M. (2006) *Pattern Recognition and Machine Learning*. New York: Springer.

Fama, E.F. (1970) 'Efficient Capital Markets: A Review of Theory and Empirical Work'. *Journal of Finance*.

Fischer, T. and Krauss, C. (2018) 'Deep learning with long short-term memory networks for financial market predictions'. *European Journal of Operational Research*.

Géron, A. (2022) *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. Sebastopol: O'Reilly Media.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E. (1991) 'Adaptive Mixtures of Local Experts'. *Neural Computation*.

Malkiel, B.G. (2019) *A Random Walk Down Wall Street*. New York: W.W. Norton & Company.

Mitchell, T.M. (1997) *Machine Learning*. New York: McGraw-Hill.

Scikit-learn (2023) *Machine Learning in Python Documentation*. Available at: https://scikit-learn.org/ (Accessed: 2026).

Sezer, O.B. and Ozbayoglu, A.M. (2020) 'Financial time series forecasting with deep learning: A systematic literature review'. *Applied Soft Computing*.

TensorFlow (2023) *TensorFlow Documentation*. Available at: https://www.tensorflow.org/ (Accessed: 2026).

Yahoo Finance (2023) *Historical Stock Price Data*. Available at: https://finance.yahoo.com/ (Accessed: 2026).

Zou, J., Zhao, Q., Jiao, Y., Cao, H., Liu, Y., Yan, Q., Abbasnejad, E., Liu, L., & Shi, J. Q. (2023). *Stock market prediction via deep learning techniques: A survey* [Preprint]. arXiv. https://arxiv.org/abs/2212.12717.