Comparison of Stock Portfolios Built from Price Prediction Models

Scott Michael Griffin Jr.*

University of Iowa

Tippie College of Business

May 11, 2024

Abstract

OLS, Decision Tree, Random Forest, KNN, SVR, and LSTM regression models for price prediction are employed to construct stock portfolios, and their subsequent performances are analyzed and compared. Portfolios for each model are constructed by predicting the future monthly price of 50 equities on the Standard & Poor's 500 index, then constructing equally weighted long positions for those with the highest predicted returns, and short positions for those with the lowest predicted returns. Portfolios are held for one month then re-balanced through the previous process until the end of the test period.

Presented to the Faculty of the Graduate School of the University of Iowa in partial fulfillment of the requirements for the degree of Master of Science in Finance

*scott-griffin@uiowa.edu

1 Introduction

Quantitative finance, also commonly referred to as mathematical finance, can be described as field of applied mathematics concerned with the modelling of financial markets and assets. Its popularity in financial research has been on the rise in recent decades, evidenced by 210,000 related papers on Google Scholar in 2023 compared to 21,200 in 1990 for the search phrase "Quantitative Finance" (see Figure 1). Studies in the field may include the development of credit risk models, volatility prediction models, asset pricing models, and future price prediction models.

This paper will focus on portfolio construction from popular asset price prediction models, including Ordinary Least-Squares (OLS) Regression, Decision Tree Regression, Random Forest Regression, K-Nearest Neighbors (KNN) Regression, Support-Vector Regression (SVR), and Long Short-Term Memory (LSTM) Recurrent Neural Networks. The popularity of all these models in financial research has increased in the past 20 years, as shown from Google Scholar search results in Figure 2.

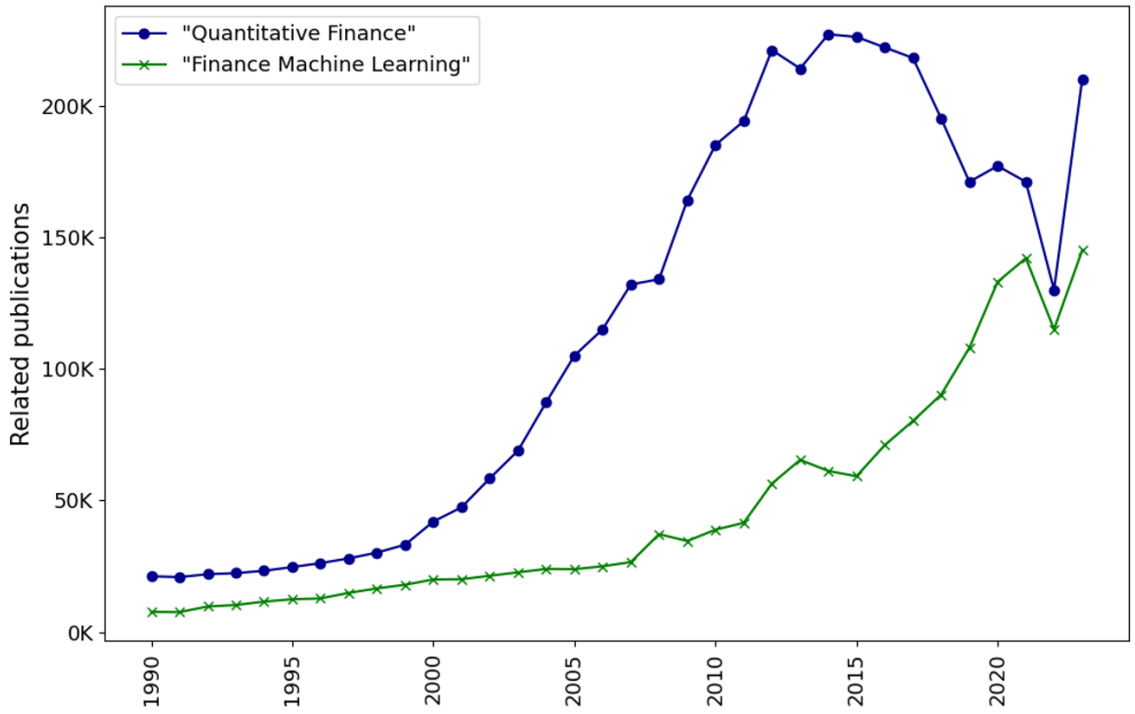

Figure 1: Publications on Google Scholar related to "Quantitative Finance" and "Finance Machine Learning"

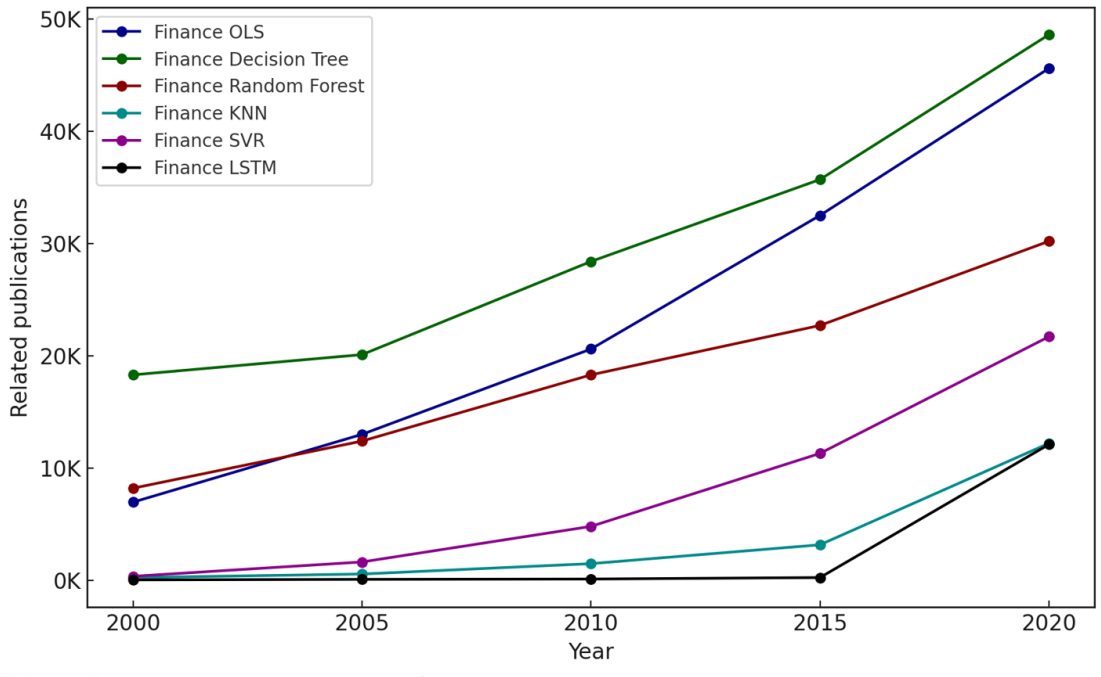

Figure 2: Publications on Google Scholar related to model keywords

2 Ordinary Least Squares (OLS) Regression

Ordinary Least Squares (OLS) Regression's discovery can be first attributed to Carl Friedrich Gauss who applies OLS to astronomic dataset in the year 1795 (Stigler, 1981). He allegedly found the concept to be too obvious to merit publication (de Prado, 2020). The term 'regression' is said to be invented in 1886 by Sir Francis Galton, a British eugenicist that applied the process to argue that hereditary human traits exhibit a 'regression' towards a mean. The term 'regression line' was invented by Karl Pearson around 1900, and mathematical proofs of regression analysis as well as maximum likelihood estimation were published by Ronald Fischer soon after (Tsay, 2013) (Stanton, 2017).

A OLS regression model is constructed as follows:

Consider a dataset comprising n observations $\{(x_i, y_i)\}_{i=1}^{n}$, where each observation i consists of

a scalar response y_i and a column vector x_i with p predictors, specifically:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{bmatrix}$$

In the context of a linear regression model, the dependent variable y_i is modeled as a linear combination of the predictors:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \varepsilon_i,$$

In vector notation:

$$y_i = x_i^T \beta + \epsilon_i,$$

where x_i is a column vector representing the i-th observation of all explanatory variables, β is a vector of p unknown parameters, and ϵ_i is a scalar denoting the unobserved random errors for the i-th observation. ϵ_i captures effects on the response y_i from factors other than the explanatory variables x_i. In matrix terms, the model is expressed as:

$$y = X\beta + \epsilon,$$

where both y and ϵ are $n \times 1$ vectors corresponding to the response variables and observation errors respectively, and X is an $n \times p$ matrix, often referred to as the design matrix, which contains the transposed vectors x_i^T as its rows, encapsulating all observations on the explanatory variables.

Typically, a constant term is incorporated in the regressor's matrix X, achieved by setting $x_{i1} = 1$ for every i. The associated coefficient β_1 is termed the intercept. Without this intercept, the model line must intersect the origin when x_i is a zero vector.

The aim in developing an OLS model is to determine each coefficient β that best satisfy the equations by solving a minimization problem (the following is a three-coefficient example, although

the number of coefficients is not limited):

$$\text{Minimize } RSS(\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2) = \sum_{i=1}^{N} u_i^2 = \sum_{i=1}^{N} \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{1,i} - \hat{\beta}_2 X_{2,i} \right)^2$$

(Goldberger, 1964) (Abbott, 2009)

3 Machine Learning Regression Models

3.1 Popularity in Finance

Machine learning tools have become increasingly popular in quantitative financial research. For the search phrase "Finance Machine Learning" there were 145,000 related papers published in 2023 compared to 7,700 in 1990 (see Figure 1). This can be theorized to be the result of increases in the granularity and number of available financial datasets (Levin, 2014) (Kolanovic, 2017). Machine learning models are better suited to represent highly-nonlinear relationships present in high-dimensional financial datasets - often unstructured - than the tradition econometric toolkit of linear and non-linear regressions and other algebraic and geometric methods (Easley, 2010).

3.2 Use in Finance

Financial researchers are still exploring potential use cases of machine learning in financial markets, as well as best practices in using them. Researchers have published papers specifically on machine learning applications in finance such as asset and index price prediction, asset volatility and liquidity prediction, anti-fraud detection, risk management, bankruptcy prediction, credit-risk assessment, and portfolio management (Ahmed, 2022). With the presentation of a promising new frontier in financial modelling, many fall prey to putting more focus on publishing models and papers than ensuring that their methods and models are statistically sound and economically useful. Many papers published describe machine learning models that can predict future asset prices and volatilities without defining clear predictive relationships between input variables and model output (a "black-box" machine learning approach). Models of this sort have limited in-

ferential information between variables and their reported successes are likely to be the result of backtest overfitting. A counter strategy to avoid these research pitfalls is to use machine learning to investigate potential predictive relationships between variables, develop causal theories between variables, and test the implications of your theory (de Prado, 2020).

3.3 Decision Tree Regression

Decision Tree models use a series of nested if-then statements related to the predictors to divide a set of data into segments. Within these segments, a model is applied to predict the outcome. In tree model terminology, the data is divided into a number of terminal nodes or leaves through multiple splits. To predict a new sample, you would follow the tree's if-then statements, using the sample's predictor values, until reaching a terminal node. A formula in that terminal node is then used to make the prediction. While the example provided illustrates a model with a simple numeric value, in other instances, the terminal node might involve a more complex function based on the predictors. The if-then statements produced by a tree specify a distinct path to a single terminal node for any given sample. A rule consists of a collection of if-then conditions (which may be formed by a tree) that are condensed into separate conditions. Rules can be simplified or pruned such that samples are covered by multiple rules. Regression trees determine (1) The predictor on which to split and the value of that split, (2) the depth or complexity of the tree, and (3) the prediction equation within the terminal nodes.

There are many techniques for constructing regression trees. One of the most utilized is the classification and regression tree (CART) methodology (Breiman, 1984). For regression, the model begins with the entire data set, S, and searches every distinct value of every predictor to find the predictor and split value that partitions the data into two groups (S_1 and S_2) such that the overall sums of squares error are minimized:

$$\text{SSE} = \sum_{i \in S_1} (y_i - \bar{y}_1)^2 + \sum_{i \in S_2} (y_i - \bar{y}_2)^2,$$

where $\bar{y}_1$ and $\bar{y}_2$ are the averages of the training set outcomes within groups S_1 and S_2, respectively.

Then within each of groups S_1 and S_2, this method searches for the predictor and split value that best reduces SSE. Because of the recursive splitting nature of regression trees, this method is also known as recursive partitioning.

After fully growing the tree, it may become excessively large, posing a risk of overfitting the training set. Consequently, the tree is *pruned* to a more manageable depth through a method known as cost-complexity tuning, as described by (Breiman, 1984).

The objective is to identify a "right-sized tree" that minimizes the error rate. This is achieved by adjusting the error rate based on the tree size:

$$\text{SSE}_{cp} = \text{SSE} + c_p \times (\#\text{ Terminal Nodes}),$$

Here, c_p denotes the *complexity parameter*. We aim to determine the smallest pruned tree that offers the lowest penalized error rate for a given c_p. (Breiman, 1984) explain the underlying theory and methods to select the optimal tree for each c_p. Similar to other regularization techniques, lower penalties often lead to more complex, and hence larger, trees. Conversely, a high complexity parameter might simplify the tree to just one split (i.e., a stump), or possibly no splits at all, indicating that no predictor sufficiently accounts for the outcome variability at that specific c_p.

To ascertain the optimal pruned tree, we assess the data across various c_p values. Each c_p produces a corresponding SSE. However, these SSE values can differ with changes in the sample of observations. (Breiman, 1984) recommend a cross-validation method to evaluate the SSE variations at each c_p. Furthermore, they advocate for applying the *one-standard-error rule* in the optimization criteria to locate the simplest tree. The target is to find the smallest tree that falls within one standard error of the tree with the smallest absolute error.

See (Kuhn, 2013) for more information about decision tree regression.

3.4 Random Forests Regression

Models based on individual trees or rules do exhibit specific limitations. Limitations include: sub-optimal predictive accuracy, and model instability, meaning minor variations in the data can significantly alter the structure of the tree or rules, thereby affecting interpretation. This reduced

accuracy stems from the models' tendency to delineate rectangular regions encompassing more uniform outcome values. If the association between predictors and responses is not effectively represented by these rectangular predictor subspaces, then tree-based or rule-based models will generally yield higher prediction errors compared to other model types. To address these issues, researchers have created ensemble methods - including random forests - that merge multiple trees (or rule-based models) into a single model. Ensemble methods typically demonstrate significantly improved predictive performance compared to individual trees, and this enhancement generally applies to rule-based models as well.

From a statistical viewpoint, one method to reduce predictor correlations is by incorporating randomness into the tree construction process. Following the introduction of bagging by Breiman, other researchers introduced modifications to infuse more randomness into the learning process. (Dietterich, 2000) proposed selecting random subsets of the top k predictors for each split in the construction of trees, a method gaining traction in bagging applications. Concurrently, approaches involving the construction of trees from entirely random subsets of descriptors were explored by (Ho, 1998). Furthering this approach, Breiman experimented with injecting noise into the response variables to disrupt the tree structure. Building upon these adaptations, he introduces a comprehensive algorithm known as random forests.

In the random forests approach, each model within the ensemble generates predictions for new samples. These predictions, totaling m from the ensemble, are then averaged to produce the forest's overall prediction. Due to the randomness in selecting predictors for each split, the correlation among trees is effectively reduced. It has been shown that a linear combination of many independent learners reduces the variance of the overall ensemble compared to any single learner within the ensemble. A random forest model accomplishes this variance reduction by incorporating strong, complex learners known for their low bias. This collection of numerous independent, robust learners leads to better error rates. Additionally, since each learner is chosen independently from the others, random forests offer strong resilience to noisy responses.

The ensemble structure of random forests obscures the direct relationship between predictors and the response. However, with trees typically serving as the base learners in this method, it

becomes feasible to assess the influence of predictors within the ensemble. Breiman initially suggested randomly permuting the values of each predictor in an out-of-bag sample for one predictor at a time per tree. The discrepancy in predictive performance between samples with and without permutation for each predictor is then collated and analyzed across the entire forest. An alternative method involves assessing the enhancement in node purity for each predictor whenever it is used, employing a specific performance metric. These individual performance gains for each predictor are subsequently compiled across the forest to ascertain the overall significance of each predictor.

See (Kuhn, 2013) for more information about random forest regression.

3.5 K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) regression algorithm forecasts a new sample using the K-closest samples from the training dataset. KNN cannot be dynamically parameterized. Instead, its construction hinges solely on the individual entries in the training data. To predict a new sample for regression, KNN identifies the nearest-K samples in the feature space. The forecast for the new sample then derives from the median, or alternatively the mean, of these K-neighbors' responses.

The fundamental KNN regression technique relies on defining the distance between two samples. The most prevalent measure used is the Euclidean distance, a direct linear distance, expressed as:

$$d_{\text{Euclidean}} = \left(\sum_{j=1}^{P} (x_{aj} - x_{bj})^2 \right)^{\frac{1}{2}},$$

where x_a and x_b represent two distinct samples.

A broader form of this is the Minkowski distance, which encompasses the Euclidean distance and is represented by:

$$d_{\text{Minkowski}} = \left(\sum_{j=1}^{P} |x_{aj} - x_{bj}|^q \right)^{\frac{1}{q}},$$

where $q > 0$. This metric simplifies into the Euclidean distance when $q = 2$, and into the Manhattan distance, also known as city-block distance, when $q = 1$, which is often used in scenarios requiring

a measure for discrete or binary predictors.

Various other distance metrics exist, suitable for specific predictor types and scientific applications. For instance, the Tanimoto, Hamming, and cosine distances offer alternatives more suitable for binary variables.

The K-Nearest Neighbors (KNN) regression method primarily relies on the computation of distances between samples, where the magnitude of the predictors significantly impacts the distance outcomes. Data characterized by predictors on diverse scales result in distances predominantly influenced by predictors with the largest scales. To mitigate this bias and ensure that each predictor equally affects the distance computation, it is advisable to normalize all predictors by centering and scaling them before applying the KNN algorithm.

Additionally, the calculation of distances can become complex if any predictor values are missing, as this makes distance computation infeasible. Under these circumstances, an analyst has several strategies: one option is to exclude samples or predictors lacking data from the analysis, though this might not always be viable especially if the missing data is substantial. An alternative method involves estimating the missing values using a simple approach such as the mean of available data or through a nearest neighbor method that leverages only the complete data.

After preprocessing the data and establishing a suitable metric for distance calculation, the next critical step involves selecting the optimal number of neighbors. Like other parameter optimization methods, the choice of K for the KNN algorithm can be ascertained through resampling techniques, ensuring robustness and accuracy in predictions compared to other models.

The basic version of K-Nearest Neighbors (KNN) is known for its simplicity and effectiveness in generating predictions, particularly when the outcomes are influenced by the proximity of local predictors. Despite some advantages, this approach faces significant challenges including high computational demands and a disparity between local structural features and the overall predictive performance of KNN. Primarily, predicting a sample requires calculating distances between the sample in question and every other sample in the dataset. This distance calculation escalates the computation time, especially as the number of samples, n, increases, necessitating that all training data be available in memory simultaneously. To address this inefficiency, an alternative

approach involves substituting the original dataset with a more memory-efficient structure that still captures the essential spatial relationships of the data points. A common solution is utilizing a data structure such as a k-dimensional tree, or k-d tree, which facilitates faster and more efficient querying of data.

3.6 Support-Vector Regression (SVR)

Suppose we have an unknown truth function $G(x)$, which depends on a vector x (defined as the *input space*). The vector $x = [x_1, x_2, \ldots, x_d]$ has d components, representing the dimensionality of the input space. The function $F(x, w)$ is a family of functions parameterized by w. The optimal w, which minimizes the error between $G(x)$ and $F(x, w)$, is determined through observation of N training instances. Our objective is to estimate w using these observations. One approach to model approximation involves representing x in a feature space z defined as

$$z = [x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d, \ldots, x_dx_1].$$

This transformation makes z a quadratic function of the input components x. Employing this feature space, the function $F_1(x, w)$ can be written as

$$F_1(x, w) = z^\top w,$$

where $F_1(x, w)$ is linear in the feature space, although it is quadratic in the input space. In general, for a polynomial of p-th order in a d-dimensional input space, the dimensionality of the feature space for vector w, denoted as f, can be calculated using the formula:

$$f = \sum_{i=d}^{p+d-1} C_{i-1},$$

where C_k represents the binomial coefficient calculated as:

$$C_k = \frac{n!}{k!(n-k)!}.$$

Another formulation was developed by (Vapnik, 1995). This method, denoted as $F_2(x, w)$, is formulated as follows:

$$F_2(x, w) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*)(v_i x + 1) + b,$$

where v_i represents a support vector corresponding to the i-th training example, and α_i, α_i^* are parameters that need optimization. This model is termed support vector regression because it incorporates training examples directly into the regression function, using the parameters α_i and α_i^* rather than just a single multiplier. The inclusion of both terms allows the model to manage various powers and cross-product terms of the input components.

Both F_1 and F_2 have a similar number of coefficients: F_1 has f free coefficients while F_2 has $2N + 1$ coefficients, determined by the N training vectors.

The loss function and the objective function in SVR can be expressed as:

$$\mathcal{J}(w) = \sum_{i=1}^{N} L(y_i - F(v_i, w)) + \lambda \|w\|^2,$$

where L is a loss function, F could be either F_1 or F_2, y_i is the observed value for the i-th example, and λ is a regularization constant. The term $\|w\|^2$ serves as a regularizer to prevent overfitting.

If the loss function is quadratic, specifically $L(\epsilon) = \epsilon^2$, and we define the function F as F_1, the optimization of the objective function can be performed using linear algebra techniques, which are well-suited to linear feature space representations. This method is commonly known as ridge regression. Consider a matrix V where each row corresponds to a training example represented in feature space, including a bias term represented as "1". Suppose N is the number of training examples, and f is the dimensionality of the feature space. Let E be a $f \times f$ diagonal matrix with each diagonal element equal to $\frac{1}{U}$, where U is a regularization parameter. Let y be an $N \times 1$ vector of observations for the dependent variable. The matrix equation for estimating $\mathbf{w}$ using ridge regression is:

$$V^\top y = [V^\top V + E]\mathbf{w}$$

The purpose of the regularization term E is to balance the mean square error against the magnitude of the $\mathbf{w}$ vector. A larger U reduces the impact of the regularization, aiming to minimize the mean

square error on the training data, potentially at the cost of generalization. The optimal value of U is determined by its performance on a validation set before being applied to a test set.

More detail on SVR is provided in (Drucker, 1997).

3.7 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) model is a variant of a Recurrent Neural Network (RNN) model designed to address the issue of vanishing gradients that plagues traditional RNNs. Its unique capability to bridge long intervals without loss of information gives it an edge over conventional RNNs, Hidden Markov Models, and other techniques for sequence learning. The primary function of LSTMs is to maintain a form of short-term memory that can extend over thousands of time steps, which is why they are termed "long short-term memory." They are effectively employed in various applications involving time series data such as handwriting recognition, speech recognition, machine translation, speech activity detection, robot control, video gaming, and healthcare.

The architecture of a typical LSTM unit includes a cell and three gates: an input gate, an output gate, and a forget gate. The cell holds values across undefined time periods while the gates manage the data flow in and out of the cell. The forget gate determines which information to retain or remove from the cell state by assigning a value between 0 and 1 to each piece of prior state information. A value close to 1 suggests retaining the information, whereas a value close to 0 indicates its removal. Similarly, the input gate decides which new data should be stored in the cell state. The output gate controls the data to be released from the current state by also assigning a value between 0 and 1, thus enabling the LSTM to retain relevant information for future predictions. This mechanism allows the network to preserve essential long-term dependencies for accurate predictions in both present and future scenarios.

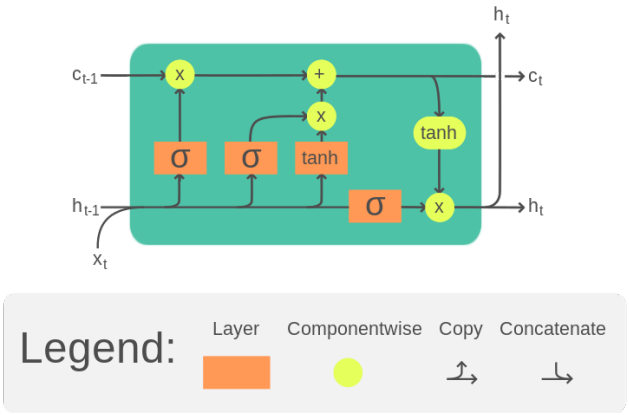

Figure 3: LSTM Cell Architecture (Source: Guillaume Chevalier)

In the equations below, the lowercase variables represent vectors. The matrices W_q and U_q contain, respectively, the weights for the input and recurrent connections. The subscript q can either be for the input gate i, output gate o, forget gate f, or the memory cell c, depending on which part of the activation is being computed. In this discussion, we use "vector notation." Thus, for instance, $c_t \in \mathbb{R}^h$ represents not just one unit of one LSTM cell, but h units of LSTM cells.

The succinct versions of the equations for the forward pass of an LSTM cell that includes a forget gate are presented below:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \sigma_h(c_t)$$

Here, the initial conditions are $c_0 = 0$ and $h_0 = 0$ and the operation $\odot$ is the Hadamard product. The subscript t indexes the time step.

The activation functions used in the model are as follows: - σ_g: The sigmoid function. - σ_c:

The hyperbolic tangent function. - σ_h: The hyperbolic tangent function, but simply the identity function, $\sigma_h(x) = x$.

An RNN equipped with LSTM units can be trained through supervised learning on sequences. This training employs an optimization method known as gradient descent, paired with backpropagation through time. This technique calculates the gradients necessary for the optimization by determining how the weights of the LSTM network should adjust based on the derivatives of the error measured at the output layer. One issue with employing gradient descent in standard RNNs is that the error gradients tend to disappear exponentially fast as the temporal gap between relevant events increases. This phenomenon is attributable to $\lim_{n\to\infty} W^n = 0$ if the spectral radius of W is less than 1. Conversely, in LSTM units, as errors are backpropagated from the output layer, the error consistently cycles through each of the LSTM unit's gates. This process, known as the "error carousel," persists until the gates adjust to appropriately modulate the error flow.

(Fischer, 2018) implements a long short-term memory (LSTM) network model for predicting directional movements of stock prices relative to the other stocks in the same index. The model predicts whether a specific stock will outperform or underperform the cross-sectional median of returns all stocks index in the subsequent time period. They test the effectiveness of this model by analyzing the performance of portfolios constructed by initiating five-day long positions in the top-k stocks in terms of relative predicted performance, and five-day short positions in the predicted worst-k stocks. They find that this strategy backtested over the period 1998 to 2015 generated an average daily return of 0.2% after transaction costs, although its a major detriment to their model's validity that they do not report the number of similar models and strategies tested before finalizing on this model. It's now well known that the probability of a spurious successful trading strategy (not based on actual financial signal but rather on noise and coincidence) increases as an increasing number of similar models are tested (de Prado, 2020).

4 Methodology

4.1 Data

Our dataset contains historical data for a 10-year period, from April 1, 2014, to April 1, 2024. With monthly frequency, it includes lagged close prices and trading volumes for 50 randomly-selected individual companies from the S&P 500 index and lagged close prices and trading volumes from the index itself. The companies available for random selection were those with 10 years of historical data. For each ticker, we store an array of close prices over the entire period, as well as a DataFrame of lagged close prices and trading volumes for that ticker and the S&P500 index.

```
{'DGX': {'y_data': array([ 61.09999847,  63.20999908,  60.68000031,  63.45999908,
        65.30999756,  67.05999756,  71.06999969,  70.13999939,
        76.84999847,  71.41999817,  75.23000336,  72.51999664,
        73.80999756,  67.80000305,  61.47000122,  67.94999695,
        68.31999969,  71.13999939,  65.66999817,  66.52999878,
        71.44999695,  75.16999817,  77.16999817,  81.41000366,
        86.36000061,  82.81999969,  84.62999725,  81.44000244,
        87.45999908,  91.90000153,  91.91999817,  97.44000244,
        98.19000244, 105.51000214, 108.76999664, 111.16000366,
       108.30999756, 108.34999847,  93.63999939,  93.77999878,
        98.45999908,  98.48999786, 105.81999969, 103.05000305,
       100.30000305, 101.19999695, 106.52999878, 109.94000244,
       107.72000122, 109.98000336, 107.91000366,  94.11000061,
        88.56999969,  83.26999664,  87.34999847,  86.55000305,
        89.91999817,  96.37999725,  95.91000366, 101.80999756,
       102.08000183, 102.37000275, 107.02999878, 101.25      ,
       106.55000305, 106.79000092, 110.66999817, 106.05999756,
        80.30000305, 110.11000061, 118.27999878, 113.95999908,
       127.06999969, 111.23999786, 114.48999786, 122.13999939,
       123.98000336, 119.16999817, 129.1499939 , 115.58999634,
       128.33999634, 131.88000488, 131.66999817, 131.97000122,
       141.80000305, 152.83000183, 145.30999756, 146.77999878,
```

Figure 4: Closing price array for the ticker DXG

For each model all tickers' data is transformed into a 70/30 train-test split. Training data for the LSTM model is organized into sets of trailing 30-month lists, reducing its training set by 30 months. Data is scaled using the default min-max scaler from sklearn.

4.2 Model Specifications

To prevent overfitting and to decrease the probability of a spurious positive result, parameter optimization is avoided. Most models use their out-of-the box parameters, or some reasonable

DGX x_data:

	Lag-1 Close	Lag-1 Volume	Lag-1 ^GSPC Close	Lag-1 ^GSPC Volume	\
3	58.689999	43848200.0	1960.229980	6.328338e+10	
4	61.099998	30378600.0	1930.670044	6.652469e+10	
5	63.209999	22622500.0	2003.369995	5.813114e+10	
6	60.680000	18803400.0	1972.290039	6.670600e+10	
7	63.459999	38149200.0	2018.050049	9.371404e+10	

	Lag-2 Close	Lag-2 Volume	Lag-2 ^GSPC Close	Lag-2 ^GSPC Volume	\
3	59.889999	36192900.0	1923.569946	6.362363e+10	
4	58.689999	43848200.0	1960.229980	6.328338e+10	
5	61.099998	30378600.0	1930.670044	6.652469e+10	
6	63.209999	22622500.0	2003.369995	5.813114e+10	
7	60.680000	18803400.0	1972.290039	6.670600e+10	

	Lag-3 Close	Lag-3 Volume	Lag-3 ^GSPC Close	Lag-3 ^GSPC Volume
3	55.930000	54231800.0	1883.949951	7.159581e+10
4	59.889999	36192900.0	1923.569946	6.362363e+10
5	58.689999	43848200.0	1960.229980	6.328338e+10
6	61.099998	30378600.0	1930.670044	6.652469e+10
7	63.209999	22622500.0	2003.369995	5.813114e+10

Figure 5: DataFrame of lagged close prices and trading volumes for ticker DGX and S&P500

parameter I picked, without trail-and-error analysis. OLS, Decision Tree, Random Forest, KNN, and SVR models were compiled, trained, and tested using the Python sklearn library. The LSTM model was built using the Python keras library.

The LSTM model specifications are from (Likhitha, 2023) which trains an LSTM model to predict Etherum prices. The reason I use their model specification is to see an actual implementation of their model, as well as to test its performance on other price series. Their LSTM model consists of a series of LSTM layers, followed by dropout layers. The first LSTM layer is made of 50 units, with subsequent layers of 60, 80, then 120 units. The Rectified Linear Unit (ReLU) activiation function is used throughout the model. The final output layer is a single neuron for predicting a single price. The Adam optimizer is used with mean squared error (MSE) as the loss function.30 periods of lookback is used for the training data and the model trains over 50 epochs with a batch size of 32.

4.3 Portfolio Construction

Let T_{train} be the training set for each asset, and T_{test} be the test set. For each model type, a model is fitted on T_{train} of each asset. We define the test set length as $|T_{\text{test}}|$.

Starting at $t = 0$, where $0 \leq t < |T_{\text{test}}|$, a prediction is generated for each asset for its price at $t =$ portfolio_horizon where for our simulation portfolio horizon was just one period (one month). Then an equally-weighted portfolio is constructed by comparing predicted returns, going long on the top-K assets and short on the bottom-K assets. For our construction process, our K was 10. The portfolio's performance is evaluated over the period $t = 0$ to $t =$ portfolio_horizon $- 1$. Then, new predictions are generated for each asset for its price at $t =$ portfolio_horizon $+$ portfolio_horizon $- 1$. Then a new portfolio is constructed and evaluated, and this process continues until $t \geq |T_{\text{test}}|$.

4.3.1 Resulting Long & Short Position Asset Frequencies

Below are tables showing the frequencies that tickers were picked for long and short positions for all models, and for each individual model. In the aggregation of all models, they generally did not identify those tickers with high mean returns to be good targets for long positions, and did not identify tickers with low mean returns to be good targets for short positions.

For the OLS model (Figure 7), stocks like UAL, COO, and DIS are commonly chosen for long positions, while WMB and BMY appear frequently in short positions. The frequency and mean return values for these selected stocks vary, indicating that the OLS model may not consistently prioritize stocks based solely on high or low mean returns when making trading decisions. The Decision Tree model (Figure 8) similarly exhibits a varied selection pattern but with different stocks highlighted. For long positions, it frequently selects UAL and KMX, whereas ORLY and BX are common choices for short positions. This model's selections also show varied mean returns, suggesting a distinct decision-making process compared to the OLS model. The Random Forest model (Figure 9) preferences are somewhat aligned with those of the Decision Tree in terms of the stocks chosen for short positions, such as BX and MSI, but differ in the long positions, choosing stocks like MMM and UAL. In the KNN model (Figure 10), DIS and MMM are frequently chosen for long positions, with MPC and ORLY often appearing in short positions. The selections for both

long and short positions include stocks with a broad range of mean returns, which is consistent across the various models, indicating no clear pattern of return-based selection. The SVR model (Figure 11) shows a preference for MMM and UAL in long positions and BX and ORLY for short positions. This model, like others, selects a mix of stocks with differing mean returns. Finally, the LSTM model (Figure 12) frequently opts for MCHP and CRL in long positions and MAR and WMB for short positions.

All Models

Long Position

	Frequency	Mean Ret.	SD Ret.
UAL	108	0.0076	0.1182
KMX	107	0.0107	0.1058
DIS	105	0.0064	0.0821
MMM	95	-0.0004	0.0639
LVS	93	0.0019	0.1003
META	72	0.0213	0.0969
MCHP	59	0.0165	0.0961
CRL	59	0.0179	0.0881
TER	58	0.0215	0.1056
AES	56	0.0054	0.0846

Short Position

	Frequency	Mean Ret.	SD Ret.
MPC	114	0.0205	0.1127
BX	107	0.0161	0.0904
MSI	106	0.017	0.0653
ORLY	103	0.0196	0.0653
ACGL	94	0.0166	0.0669
WMB	80	0.0016	0.0967
HAL	72	0.0059	0.1427
PXD	72	0.0073	0.1056
HUBB	60	0.0138	0.0746
MSCI	52	0.0246	0.0736

Figure 6: Long and short position frequencies for all models

OLS

Long Position

	Frequency	Mean Ret.	SD Ret.
UAL	27	0.0076	0.1182
COO	20	0.0103	0.0673
DIS	15	0.0064	0.0821
LVS	15	0.0019	0.1003
MCHP	14	0.0165	0.0961
ABT	14	0.0103	0.058
DECK	11	0.0248	0.0915
KMX	11	0.0107	0.1058
DG	11	0.0117	0.0728
MAR	10	0.0162	0.0929

Short Position

	Frequency	Mean Ret.	SD Ret.
WMB	23	0.0016	0.0967
BMY	23	0.0028	0.066
MPC	21	0.0205	0.1127
HAL	21	0.0059	0.1427
HUBB	17	0.0138	0.0746
AVB	15	0.0037	0.059
SPG	14	0.0046	0.0947
DGX	11	0.0094	0.0731
ACGL	10	0.0166	0.0669
BX	10	0.0161	0.0904

Figure 7: OLS model long and short position frequencies

Decision Tree

Long Position

	Frequency	Mean Ret.	SD Ret.
UAL	22	0.0076	0.1182
KMX	21	0.0107	0.1058
LVS	17	0.0019	0.1003
MMM	16	-0.0004	0.0639
AES	14	0.0054	0.0846
PH	14	0.0171	0.0834
TER	13	0.0215	0.1056
META	12	0.0213	0.0969
DD	10	0.0044	0.0894
ABT	9	0.0103	0.058

Short Position

	Frequency	Mean Ret.	SD Ret.
ORLY	25	0.0196	0.0653
BX	22	0.0161	0.0904
MSI	19	0.017	0.0653
ACGL	17	0.0166	0.0669
DGX	14	0.0094	0.0731
MPC	13	0.0205	0.1127
HUBB	12	0.0138	0.0746
SHW	10	0.0167	0.0721
DRI	10	0.0159	0.0858
MSCI	10	0.0246	0.0736

Figure 8: Decision Tree model long and short position frequencies

Random Forest

Long Position

	Frequency	Mean Ret.	SD Ret.
MMM	22	-0.0004	0.0639
UAL	20	0.0076	0.1182
DIS	19	0.0064	0.0821
LVS	19	0.0019	0.1003
KMX	18	0.0107	0.1058
DVA	17	0.0096	0.0886
TER	13	0.0215	0.1056
AES	12	0.0054	0.0846
CE	9	0.0129	0.0832
MKC	9	0.0091	0.0589

Short Position

	Frequency	Mean Ret.	SD Ret.
BX	26	0.0161	0.0904
MSI	26	0.017	0.0653
ORLY	24	0.0196	0.0653
ACGL	15	0.0166	0.0669
MSCI	14	0.0246	0.0736
MPC	13	0.0205	0.1127
PXD	12	0.0073	0.1056
DECK	11	0.0248	0.0915
CRL	11	0.0179	0.0881
MAR	10	0.0162	0.0929

Figure 9: Random Forest model long and short position frequencies

KNN

Long Position

	Frequency	Mean Ret.	SD Ret.
DIS	23	0.0064	0.0821
MMM	21	-0.0004	0.0639
LVS	19	0.0019	0.1003
DVA	16	0.0096	0.0886
KMX	15	0.0107	0.1058
UAL	14	0.0076	0.1182
AES	12	0.0054	0.0846
META	12	0.0213	0.0969
MKC	11	0.0091	0.0589
CRL	10	0.0179	0.0881

Short Position

	Frequency	Mean Ret.	SD Ret.
MPC	26	0.0205	0.1127
ORLY	24	0.0196	0.0653
BX	23	0.0161	0.0904
MSI	22	0.017	0.0653
PXD	19	0.0073	0.1056
HAL	18	0.0059	0.1427
ACGL	17	0.0166	0.0669
WMB	16	0.0016	0.0967
DECK	11	0.0248	0.0915
HUBB	10	0.0138	0.0746

Figure 10: KNN model long and short position frequencies

SVR

Long Position

	Frequency	Mean Ret.	SD Ret.
MMM	27	-0.0004	0.0639
UAL	24	0.0076	0.1182
LVS	22	0.0019	0.1003
DD	21	0.0044	0.0894
DIS	19	0.0064	0.0821
KMX	15	0.0107	0.1058
WMB	13	0.0016	0.0967
FE	12	0.0036	0.0588
DVA	12	0.0096	0.0886
META	10	0.0213	0.0969

Short Position

	Frequency	Mean Ret.	SD Ret.
BX	26	0.0161	0.0904
ORLY	24	0.0196	0.0653
MSI	24	0.017	0.0653
DECK	20	0.0248	0.0915
MSCI	19	0.0246	0.0736
CMG	16	0.0173	0.0982
MPC	15	0.0205	0.1127
ISRG	15	0.0213	0.083
ACGL	14	0.0166	0.0669
CRL	13	0.0179	0.0881

Figure 11: SVR model long and short position frequencies

LSTM

Long Position

	Frequency	Mean Ret.	SD Ret.
MCHP	27	0.0165	0.0961
CRL	27	0.0179	0.0881
KMX	27	0.0107	0.1058
PH	26	0.0171	0.0834
TGT	26	0.0133	0.0879
MSCI	25	0.0246	0.0736
DIS	24	0.0064	0.0821
BX	23	0.0161	0.0904
META	20	0.0213	0.0969
AES	14	0.0054	0.0846

Short Position

	Frequency	Mean Ret.	SD Ret.
MAR	27	0.0162	0.0929
WMB	26	0.0016	0.0967
MPC	26	0.0205	0.1127
HAL	23	0.0059	0.1427
PXD	22	0.0073	0.1056
ACGL	21	0.0166	0.0669
CHRW	17	0.003	0.0624
DRI	16	0.0159	0.0858
BKNG	16	0.0131	0.0871
BMY	13	0.0028	0.066

Figure 12: LSTM model long and short position frequencies

5 Portfolio Performance Comparisons

The provided figures present a detailed comparison of the performance of portfolios constructed using predictions from various models over a specific time period. The line graph in Figure 13 tracks the progression of portfolio values, illustrating how each model's predictive accuracy affects investment returns over time. Notably, none of the models exhibit exceptional performance; however, some show relative predictive competence.

The Ordinary Least Squares (OLS) model, for example, appears to lead in portfolio value growth, peaking around 1.3. While this suggests some level of predictive accuracy, it's important to note that the graph alone might not fully represent risk-adjusted returns, as further detailed in Figure 14. The LSTM model displays an opposite trend, with generally declining portfolio values, indicating less effectiveness in market prediction within this context.

The Random Forest and Decision Tree models perform relatively well, demonstrating a generally upward, albeit volatile, trajectory. This pattern suggests these models are somewhat sensitive to market conditions but can still provide useful predictions. The Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) models exhibit moderate performance, with their portfolio values oscillating around the initial value and closing slightly higher.

Further insights are provided in Figure 14, which summarizes key statistical metrics including

mean return, standard deviation of returns, Sharpe Ratio, and total return. The Random Forest model, while having the highest total return among the models, also maintains a modest Sharpe Ratio, indicating a reasonable balance between return and risk. Similarly, the Decision Tree model shows comparable performance in terms of total return and Sharpe Ratio.

Conversely, the OLS model, despite its apparent strong performance on the graph, exhibits a negative mean return and a negative Sharpe Ratio, which flags potential high risk or inconsistent performance. The SVM and KNN models achieve only modest total returns with relatively lower Sharpe Ratios, indicating a higher level of risk or less effective risk management compared to Random Forest and Decision Tree models. The LSTM model, with the lowest total return and Sharpe Ratio, appears least suited for this particular predictive task.

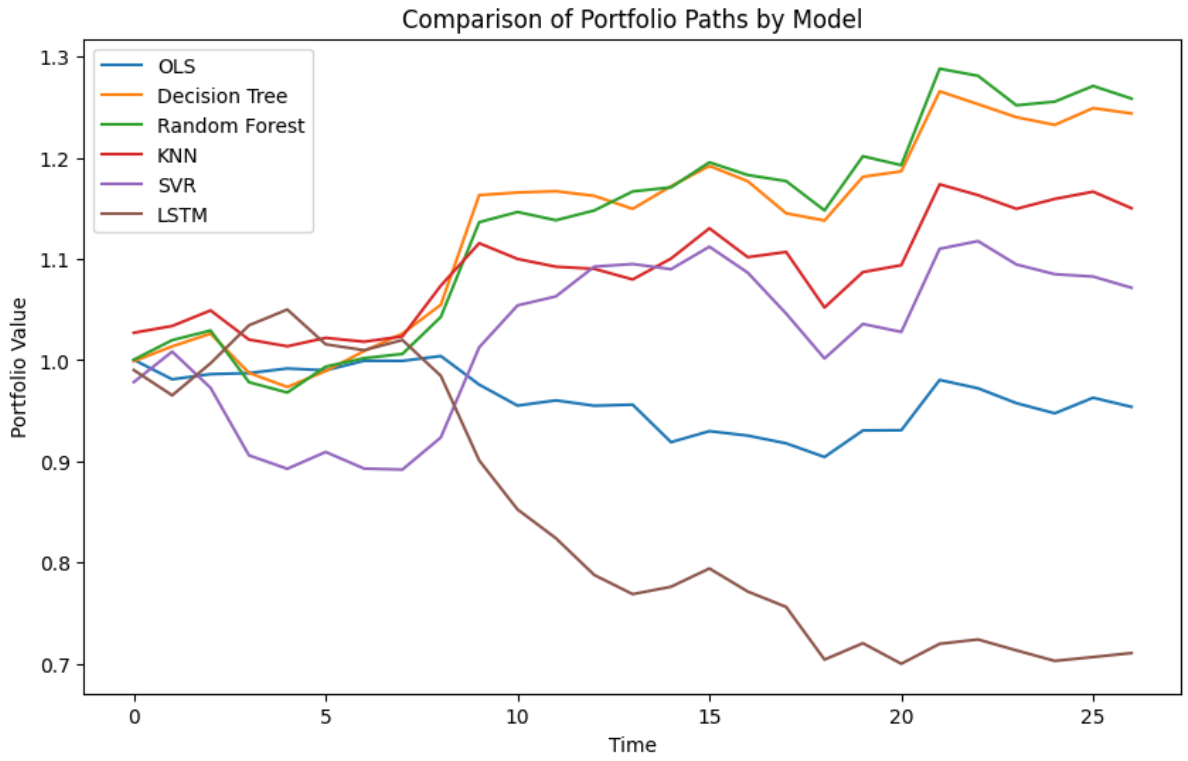

Figure 13: Comparison of portfolio paths by model

Model	Mean Ret.	SD Ret.	Sharpe Ratio	Total Ret.
OLS	-0.0017	0.0177	-0.1	0.95
Decision Tree	0.0088	0.028	0.31	1.25
Random Forest	0.0093	0.0291	0.32	1.26
KNN	0.0047	0.0249	0.19	1.12
SVR	0.0041	0.0355	0.12	1.1
LSTM	-0.0122	0.0309	-0.39	0.72

Figure 14: Model portfolios summary

5.1 Predicted vs. Realized Returns by Model

The following analysis of the predictive performance of various models in forecasting returns demonstrates a consistent trend of divergence between predicted and realized returns. In the case of the Ordinary Least Squares (OLS) model, as shown in Figure 15, there is a notable discrepancy where the predicted returns generally maintain a higher and less volatile trajectory compared to the realized returns, which exhibit greater fluctuation and lower values overall. This pattern suggests that the OLS model may be overestimating returns, indicating a possible lack of responsiveness to market volatility.

Similar observations can be made for the Decision Tree model depicted in Figure 16, where predicted returns also fail to accurately mirror the market's actual movements. The predictions here are smoother and consistently higher than the realized returns, pointing towards potential overfitting issues where the model may be capturing noise rather than underlying market trends.

The Random Forest model, illustrated in Figure 17, displays a slightly better alignment in the direction of trends compared to the previous models. However, it too tends to overestimate the magnitude of returns, particularly in the latter half of the observed period. This consistent overestimation across earlier and later stages suggests that while the model can capture the general direction of market movements, it struggles with accurately predicting their intensity.

Figure 18 presents the K-Nearest Neighbors (KNN) model, which shows one of the most significant disparities between predicted and realized returns, with predictions invariably higher than actual outcomes. This consistent overestimation across the timeline raises concerns about the model's ability to adjust dynamically to changing market conditions, potentially due to its re-

liance on local similarities which may not be indicative of broader market trends.

Support Vector Regression (SVR), seen in Figure 19, moderately aligns with the overall trend of the realized returns but lacks precision in matching the peaks and troughs, resulting in a smoother prediction curve. The SVR model's tendency to produce less volatile predictions might be limiting its effectiveness in scenarios where market conditions are highly unpredictable or subject to sudden changes.

Lastly, the Long Short-Term Memory (LSTM) model, as shown in Figure 20, exhibits the most considerable divergence of all the models tested. The predicted returns follow a markedly stable and ascending path, in stark contrast to the much lower and nearly flat line of realized returns. This severe mismatch suggests profound overfitting or a fundamental misalignment with the dynamics governing market behavior, indicating a critical need for reevaluation of the model's parameters or training process.

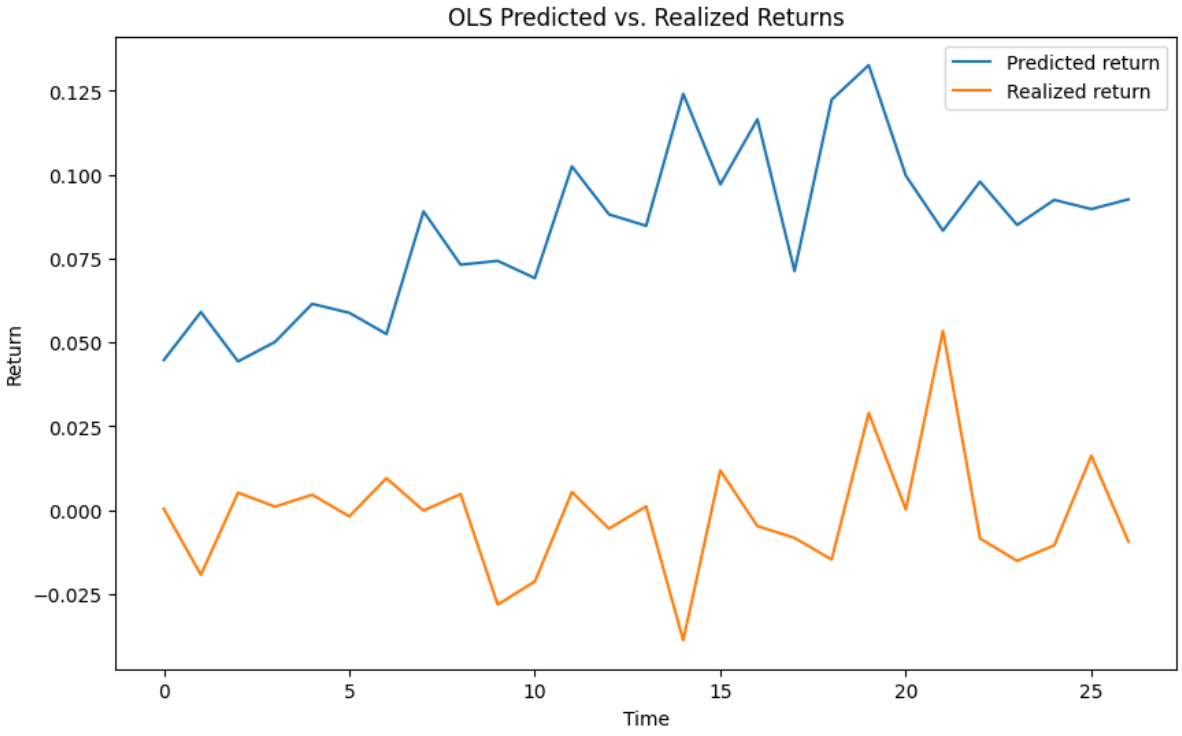

Figure 15: OLS predicted vs. realized returns

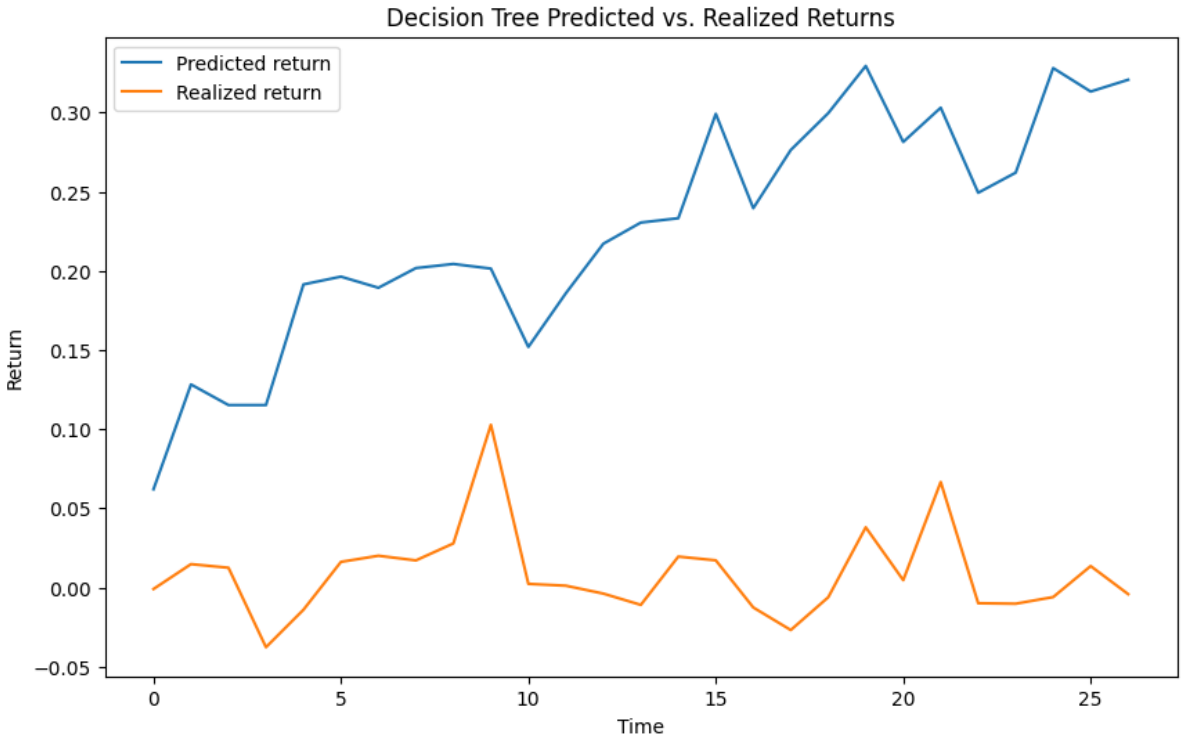

Figure 16: Decision Tree predicted vs. realized returns

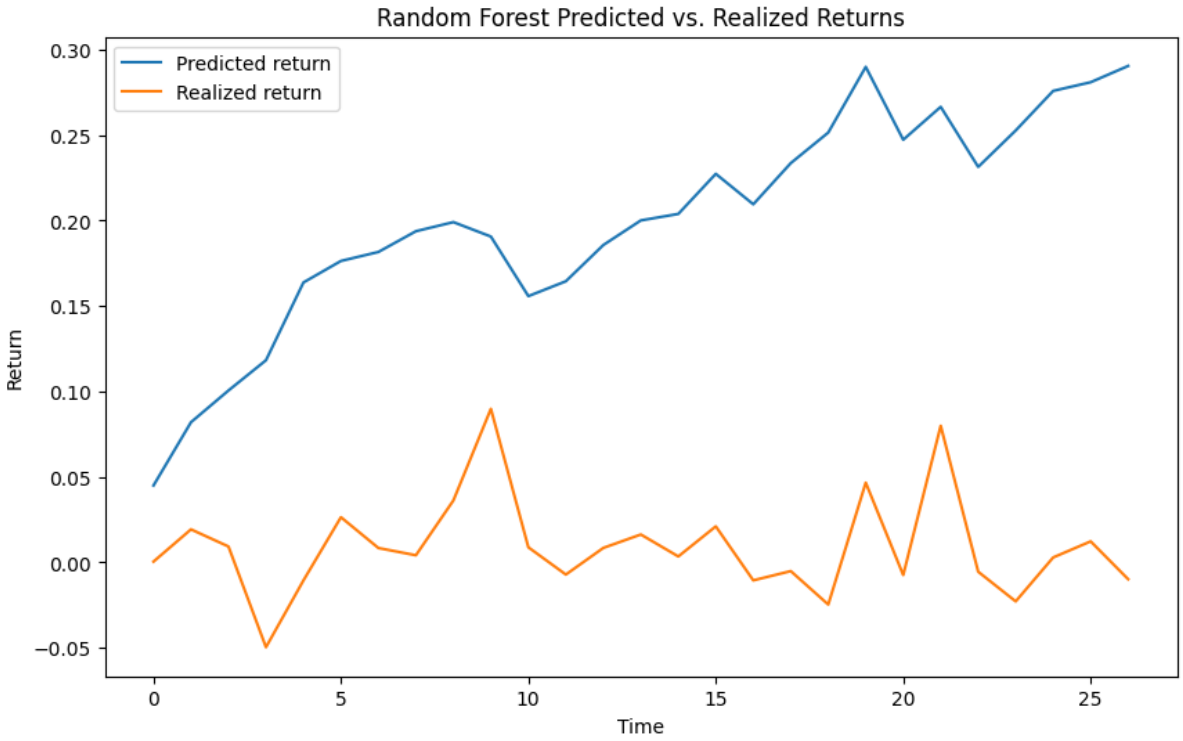

Figure 17: Random Forest predicted vs. realized returns

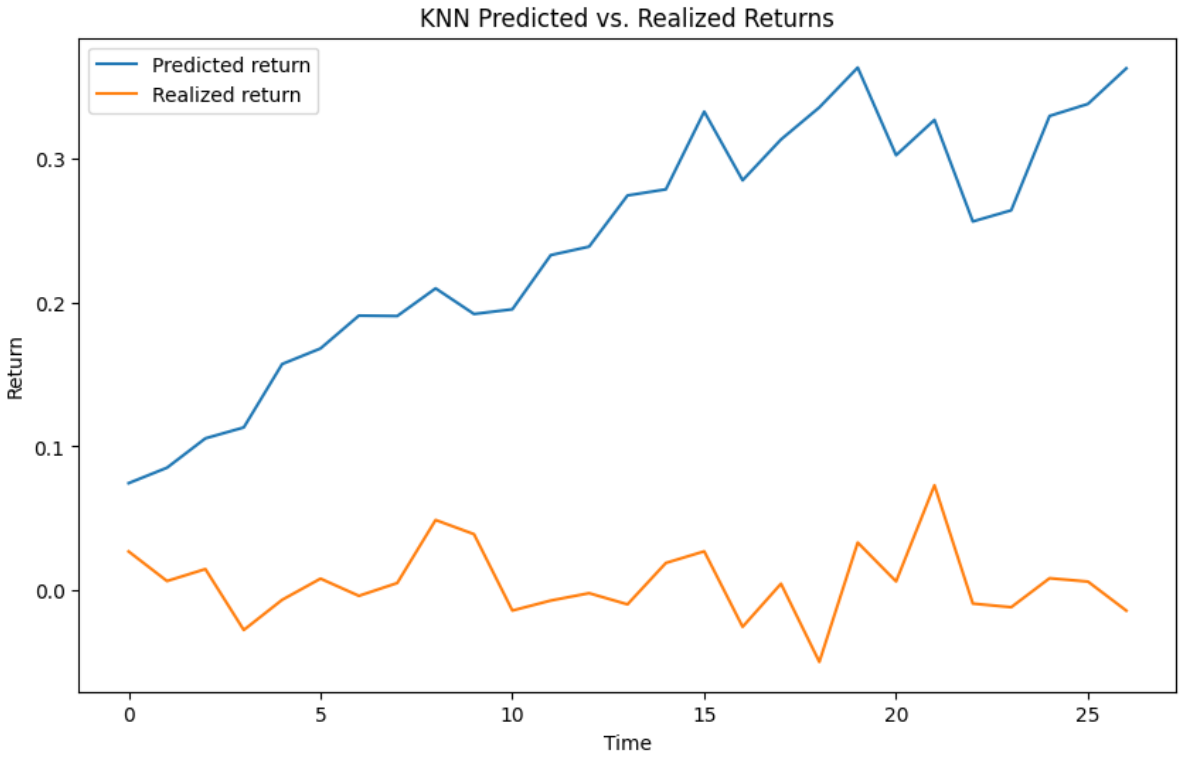

Figure 18: KNN predicted vs. realized returns

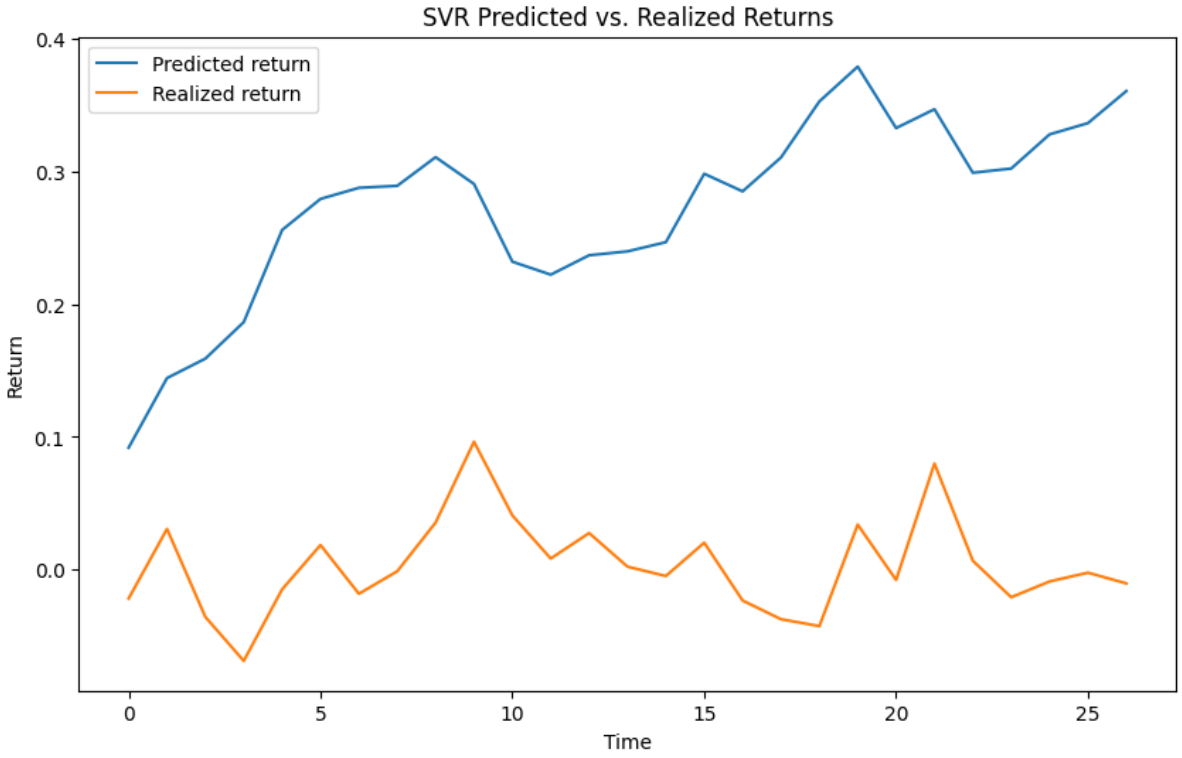

Figure 19: SVR predicted vs. realized returns

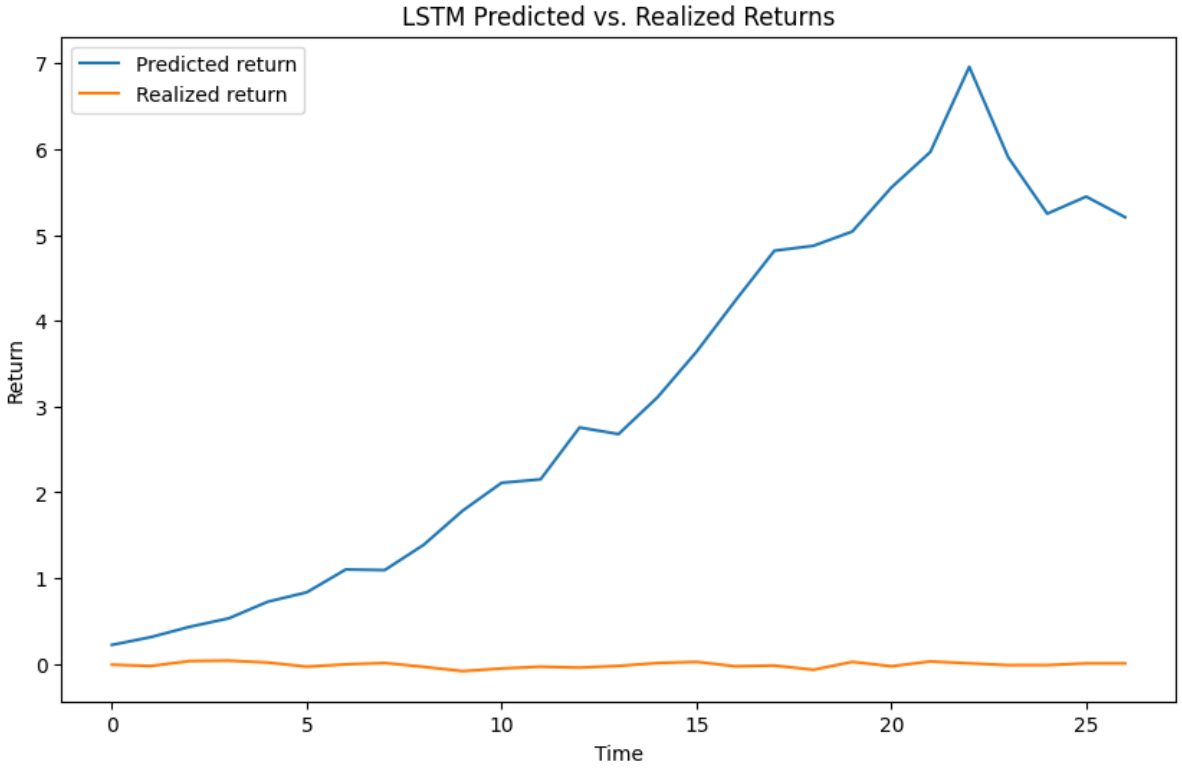

Figure 20: LSTM predicted vs. realized returns

6 Conclusion

In this paper we investigated the effectiveness of various predictive models—Ordinary Least Squares (OLS), Decision Tree, Random Forest, K-Nearest Neighbors (KNN), Support Vector Regression (SVR), and Long Short-Term Memory (LSTM)—in constructing stock portfolios over a 10-year period using data from the S&P 500 index. Utilizing a methodology that involved predicting the monthly prices of 50 randomly selected equities and constructing portfolios by taking long positions in assets with the highest predicted returns and short positions in those with the lowest, we assessed each model's performance.

The OLS model demonstrated significant portfolio value growth but suffered from a negative mean return and Sharpe Ratio, suggesting high risk or inconsistency. The LSTM model, despite its advanced temporal capabilities, showed a large decline in portfolio values. The Random Forest and Decision Tree models exhibited upward but volatile trajectories, and the SVM and KNN models achieved moderate performance with stable values, though their lower Sharpe Ratios pointed to

increased risks or suboptimal risk management.

Our results do underscore the complex nature of financial market prediction and the need for careful model selection based on causal relationships between variables, and causal theories. The varying performances across models highlight that while some can capture certain market dynamics effectively with price alone, none are able to universally guide predictions in a confident fashion in all conditions. While time-series price data may provide some insight into the movement of stock prices, this approach is certainly flawed.

References

Abbott, M. (2009). Ols estimation of the multiple (three-variable) linear regression model.

Ahmed. (2022). Artificial intelligence and machine learning in finance: A bibliometric review. *Research in International Business and Finance*.

Breiman. (1984). Classification and regression trees.

de Prado, L. (2020). Interpretable machine learning: Shapley values.

Dietterich. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems*.

Drucker. (1997). Support vector regression machines. *Advances in Neural Information Processing Systems*.

Easley. (2010). Networks, crowds, and markets: Reasoning about a highly connected world.

Fischer. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*.

Goldberger. (1964). Classical linear regression.

Ho. (1998). The random subspace method for constructing decision forests. *Trans. on Pattern Analysis and Machine Intelligence*.

Kolanovic. (2017). Big data and ai strategies: Machine learning and alternative data approach to investing. *J.P. Morgan Quantitative and Derivative Strategy*.

Kuhn. (2013). Applied predictive modeling.

Levin, E. . (2014). Economics in the age of big data. *Science*, *346*.

Likhitha. (2023). Unveiling ethereum's future: Lstm-based price prediction and a systematic blockchain analysis.

Stanton. (2017). Galton, pearson, and the peas: A brief history of linear regression for statistics instructors.

Stigler. (1981). Gauss and the invention of least squares. *Annals of Statistics*, *9*.

Tsay. (2013). Multivariate time series analysis: With r and financial applications.

Vapnik. (1995). Support-vector networks. *Machine Learning*, *20*.