Support Vector Regression for Stock Price Predicition

Alexander DeLise

April 23, 2024

Abstract

This paper explores support vector machines (SVMs), and thus support vector regression (SVR) for predicting stock prices during a forecasting period of 14 days. Utilizing historical stock price data from three companies (AAPL, COKE, and GOOG), we investigate the effectiveness of different kernel functions in SVR, namely linear, radial basis function (RBF), and polynomial kernels. Through empirical validation, we assess the predictive accuracy of each kernel and identify key insights into their performance. We also compare the results to stock price prediction using a deep neural network (DNN), an already-proven regression method. Our results indicate that the linear kernel outperforms the RBF and polynomial kernels in terms of mean ℓ_2-loss, demonstrating its suitability for short-term stock price forecasting. However, the study acknowledges several limitations, including the narrow scope of analysis, reliance on empirical parameter tuning, and the exclusive focus on mean ℓ_2-loss as the evaluation metric. Recommendations for future research include expanding the dataset to include a more diverse set of stocks, exploring advanced parameter tuning techniques, and investigating alternative evaluation metrics for model assessment. Overall, this study provides valuable insights into the application of SVR for stock price prediction and highlights areas for further investigation to enhance predictive accuracy and robustness.

1 Introduction

The stock market is a historically unpredictable environment, though economists and mathematicians often attempt to uncover hidden patterns within its history to forecast its future activity. The most crucial job component of investors is the selection of stocks that will maximize their profits a given period, i.e. those that will outperform the market in that period [1]. Due to the stock market being nonlinear, nonparametric, noisy, and deterministically chaotic [2], selecting such stocks proves immensely difficult.

One technique to aid in the stock selection process is the use of support vector machines (SVMs) and thus support vector regression (SVR) for stock price forcasting. The basic training principle of SVMs is that their expected classification (generalization) error is minimzed so that they create an effective predictive model [3].Training data for such SVMs usually originates in databases of stock prices and indicators [1]. In general, SVM training involves solving convex programming problems that require numerous constraints, and thus many computational resources [3]. The goal of an SVM for stock classification is to construct an optimal separating hyperplane within the training data set using a linearly constrained quadratic programming problem [4]. To aid in the classification of data, kernel methods are a popular choice among scientists due to their robust capabilities. One key benefit of SVMs is that their solution is always unique and globally optimal, contributing to their reliability and robustness.

The structure of the paper will first introduce the problem of stock classification and then describe the basic theory behind SVMs from [5]. The subsequent sections will outline the problem formulation and data accumulation, with experimental results and analyses following. Finally, the results from the SVR will be compared to those from a Deep Neural Network (DNN). The primary goal of the paper is to demonstrate an example of a practical application of SVMs in stock selection and evaluating their performance compared to traditional models.

2 Description of the Mathematics

The application of SVMs allows for the creation of a support vector regression, the tool that will be used to predict the future prices of stocks. To understand the basic theory behind SVRs, it is first beneficial to grasp a solid foundation of SVM theory. We will build a strong intution into how SVMs classify data and how kernelized methods allow for the classification of complex, noisy, and otherwise nonlinear data. With this background knowledge, we will provide a brief introduction into the SVR process.

2.1 Support Vector Machines

2.1.1 Linearly Separable Data

It is essential to begin with the simple linear classification case with linearly separable data, with the majority of its derivation arriving from [5] and [6]. The premise of of linear binary classification problems involves an input space $\boldsymbol{\mathcal{X}} = \mathbb{R}^n$ and an output space $\boldsymbol{\mathcal{Y}} = \{-1, 1\}$. We consider a hypothesis set that consists of linear classification functions that is defined as

$$H = \{\boldsymbol{x} \mapsto \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b) : \boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}\}.$$

Given a collection of datapoints, the hypothesis $\boldsymbol{x} \mapsto \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$ labels all instances of $\boldsymbol{x}$ on one side of the dividing hyperplane $\boldsymbol{w}^T\boldsymbol{x}+b$ positively, and all points on the other side negatively. When data is linearly separable, there exists infinitely many hyperplanes that separate the data, thus the goal of SVMs is to select the hyperplane that is of maximum distance to the closest points in a given sample of data $S = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ with labels $\boldsymbol{y} = (y_1, \ldots, y_m)$. The hyperplane that satisfies this condition is the maximum margin, and can be interpreted as the safest choice since it requires the largest perturbation of an input $\boldsymbol{x}$ to change its label. Together, this basic information is summarized in the following figure:

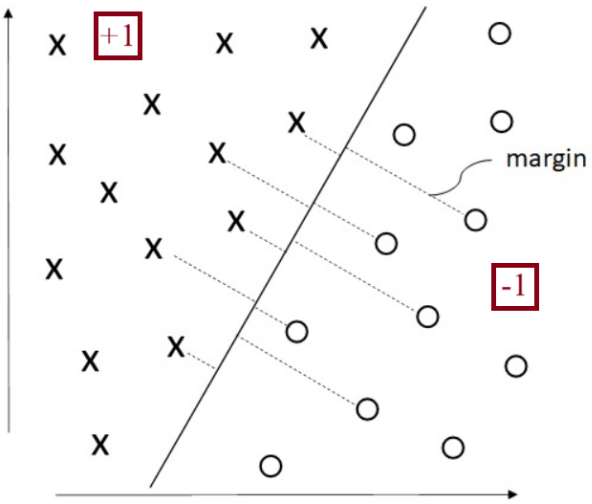

Figure 1: Linearly Separated Data with Labels and Margins

The maximal margin hyperplane, the safest choice, is precisely the solution of the convex optimation problem (the primal problem)

$$\min_{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}} \frac{1}{2}||\boldsymbol{w}||_2^2 \quad \text{subject to} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 \quad \text{for} \quad i = 1, \ldots, m. \tag{1}$$

However, to solve this problem, it is necessary to introduce Lagrange multipliers and the famous Karush-Kuhn-Tucker (KKT) conditions. See [5] for the full derivation, which eventually results in the transformed

convex optimization problem (called the dual problem)

$$\begin{aligned} \text{maximize} \quad & W(\alpha) = -\sum_{i=1}^{m} \alpha_i + \frac{1}{2}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\ \text{subject to} \quad & a_i \geq 0, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{2}$$

The dual problem has simpler constraints than the primal problem, thus making it easier to solve numerically. Another benefit of solving the dual problem is that the number of support vectors is typically smaller than the sample size, and that they are sparse. Finally, the solution of the linear SVM depends only on the inner product between two vectors $\boldsymbol{x}_i^T \boldsymbol{x}_j$ and not on the vectors themselves. This allows SVMs to be kernelized, which we will explore further.

2.1.2 General Kernel Theory

In practice, data is almost never linearly separable, thus nonlinear methods are essential to generalize the classification of various types of high dimensional data. By "pre-processing" our data, we can transform the data in a such way that the SVM classification problem is transformed so that a simpler hyperplane can be found [7]. More specifically, we seek a mapping $\boldsymbol{z} = \phi(\boldsymbol{x})$ that transforms a d-dimensional input vector $\boldsymbol{x}$ into a d' dimensional vector $\boldsymbol{z}$ where typically $d' >> d$. The goal of the new mapping is to chose a ϕ such that the new training data $\{\phi(\boldsymbol{x}_i), y_i\}$ is separable by a hyperplane.

Determining the optimal mapping ϕ is difficult, especially if one were to construct one explicitely for each dataset. Fortunately, if $\phi(\boldsymbol{x})$ casts an input vector into a high enough space, that is $d' >> d$, the data should become separable [7]. This also creates a burden computationally as the construction of a higher dimensional matrix becomes prohibitive. Also, increasing the complexity of the problem into higher dimensional spaces can lead to overfitting [7].

A key benefit of SVMs due to their very nature is the avoidance of many of the aforementioned problems. Given a mapping $\boldsymbol{z} = \phi(x)$, we replace all occurances of $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$, thus transforming our original optimization problem into

$$\begin{aligned} \text{maximize} \quad & W(\alpha) = -\sum_{i=1}^{m} \alpha_i + \frac{1}{2}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \\ \text{subject to} \quad & a_i \geq 0, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned} \tag{3}$$

In this scenario we work only with the dot products between two mappings of vectors $\phi(\boldsymbol{x}_1)$ and $\phi(\boldsymbol{x}_2)$. If one knows the formula for the dot product in the higher dimensional feature space, namely the kernel, written as

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2),$$

we never have to deal directly with the mapping $\boldsymbol{z}$ [7]. By assembling the problem in this manner, we determine the optimal separating hyperplane as usual, except now this hyperplane is in some unknown feature space. Although it is tedious to design a different feature mapping ϕ for each data set, there exist certain kernels that have already been discovered, such as the two mentioned in this paper.

2.1.3 Radial Basis Function

The Radial Basis Function (RBF) kernel is among the most popular forms of kernelization due to its similarity to the Gaussian distribution [8]. The RBF kernel function takes two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and computes their similarity, i.e. how close they are to each other. Mathematically, the kernal is represented as

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(\frac{-||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2^2}{2\sigma^2}\right) = \exp\left(-\gamma||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2^2\right) \tag{4}$$

where σ is the variance of the data and the hyperparameter $||\boldsymbol{x}_1 - \boldsymbol{x}_2||$ is the ℓ_2-norm between the two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Using the RBF kernel gives the classifier

$$\begin{aligned} f(x) &= \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\boldsymbol{x}_1, \boldsymbol{x}_2) + b\right) \\ &= \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \exp\left(-\gamma ||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2^2\right) + b\right) \end{aligned} \tag{5}$$

Here, the RBF kernelized SVM determines the number and location of centers needed to form an RBF network with the highest expected generalization performance [7] with the desire of the predicted stock price having a low ℓ_2-loss.

Since the γ parameter defines the width of the RBF kernel, it also controls the influence of individual samples on the decision boundary, i.e. it controls the trade-off between minimizing the training error and maximizing the margin of the decision boundary in feature space. Smaller values of the γ parameter impose a softer margin, allowing for more training points to be misclassified and thus a lower risk of overfitting as it improves generalization. Conversely, a larger value of the γ parameter imposes a harder margin, leading to fewer training errors but potentially sacrificing generalization performance (higher risk of overfitting). Figure 2 summarizes these results.

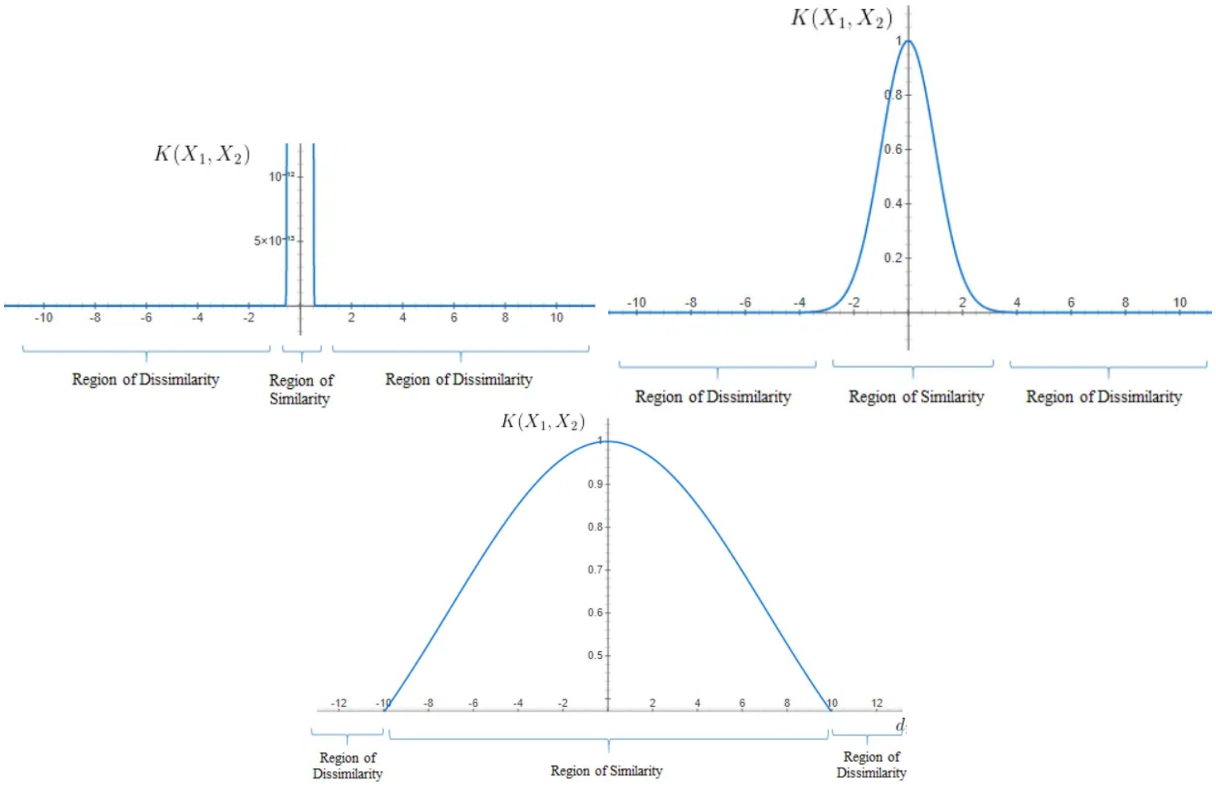

Figure 2: Regions of Similarity for $\gamma = 10, 1, 0.1$ [8]

2.1.4 Polynomial Kernel

The polynomial kernel represents the similarity of two training vectors in a feature space over polynomials of the orginal variables, thus permitting the learning of nonlinear models. We note that much of the theory behind polynomial kernels in this paper come from [9]. A polynomial kernel is of the form

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\gamma \boldsymbol{x}_1^T \boldsymbol{x}_2 + r)^d, \tag{6}$$

where γ and r are parameters and d is the degree. The parameter γ, like that found in the RBF kernel, defines how much influence a single training example has, thus determining the nonlinearity of the decision boundary. Large values of γ makes the model more sensitive to the training data, thus possibly leading to overfitting, and smaller values values make the model less sensitive, leading to underfitting. The constant r shifts the origin of the polynomial function, allowing for datasets that are not centered around the origin. The degree d of the polynomial determines the dimension of the mapping of the feature space. Larger values of d create a more complex decision boundary, though one that might not be representative of the data, i.e. larger values of d do not neccessarily correlate to more accurate classification.

2.2 Support Vector Regression

With a key intuition into how SVMs and their kernelized versions classify data, we introduce SVR to generalize the process to yet-to-be-seen data. Much like SVMs, SVR is charactarized by the use of kernels, the number of support vectors, sparse solution, etc. and have been proven to be an effective tool in real-value function estimation. We develop a short introdcution into SVR theory based on the work of [10].

The SVR process trains using a symmetrical loss function by using a supervised learning approach and penalizing high and low misestimates of data. A flexible tube of minimal radius is formed symmetrically around the estimated function such that the absolute values of the errors that are less than a certrain threshold ϵ are ignored both above and below the estimate. This approach requires that datapoints outside of the tube are penalized and those inside are not. Since SVR does not depend on the dimensionality of the input space, kernel methods are often utilized since they do not affect the computational compexity of the result. If using the proper kernel for a problem, one can expect high predicition accuracy.

More mathematically, SVR is formulated as an optimization problem by defining convex ϵ-sensitive loss function to be minimized and subsequently determining the flattest tube that contains a majoirty of the input data. From there a multi-objective function is constructed from the loss function and the geometric properites of the tube, which has a unique solution. The resulting function, assuming the input data is linear, takes the form

$$f(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}^T \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \boldsymbol{w}^T \boldsymbol{x} + b. \tag{7}$$

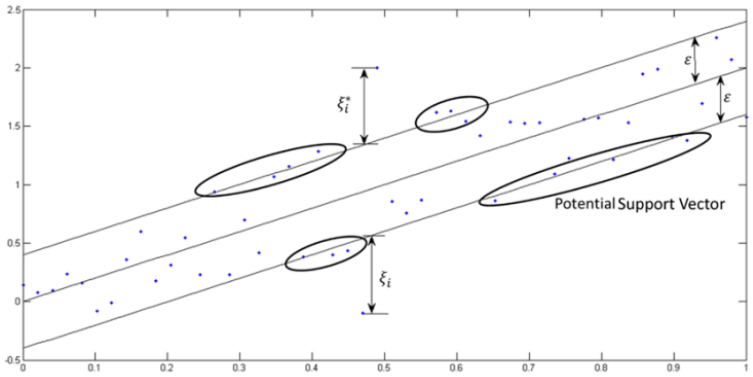

Figure 3: Linear SVR Regression [10]

Assuming nonlinear input data, we also introduct kernel methods for SVRs using a feature mapping ϕ. Using the dual-problem representation of the objective function yields

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{8}$$

where α and α^* are Lagrange multipliers of the dual problem, and $k(\boldsymbol{x}_i, \boldsymbol{x})$ is the kernelization

$$k(\boldsymbol{x}_i, \boldsymbol{x}) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}).$$

Like with SVMs, many popular kenerls exist like RBF and polynomial kernels. Although we do not explicitely derive the classfier functions for SVR, it is easy to see their direct relationship with their SVM counterparts. Figure 4 displays an example of a nonlinear SVR.

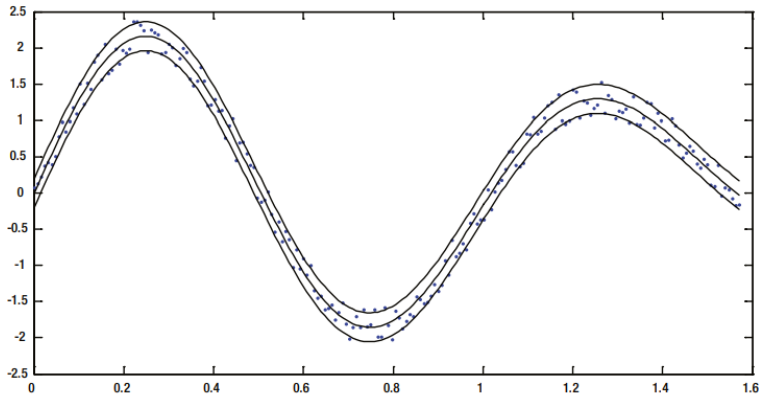

Figure 4: Linear SVR Regression [10]

3 Methods

3.1 Problem Formulation

Given the daily closing price of a stock, we aim to determine the best SVR regression method to forecast its future stock price in terms of least mean ℓ_2-loss. The prediction of stock prices poses a challenging task due to the complex and nonlinear nature of financial markets, characterized by various influencing factors and uncertainties. By leveraging the capabilities of the kernelized SVR, we seek to capture intricate patterns and relationships inherent in historical stock price data to make accurate predictions for a defined forecasting horizon.

3.2 Data Collection and Feature Selection

Stock price data is obtained from Yahoo Finance using the `yfinance` package in Python. The `yfinance` package allows users to access and download historical market data, including stock price data, from Yahoo Finance. We obtain data for the AAPL, GOOG, and COKE stocks. The head of the AAPL dataset from is displayed in Table 1. The data spans from January 1, 2010, to December 31, 2016. Although there are a few missing days of data per year, there is ample data to train and test the kernelized SVRs.

3.3 Model Specification

We test three different kernelizations for the SVR: the linear, RBF, and polynomial kernelizations. For the linear kernelization, there are no parameters to tune. To tune the parameters of the other kernelizations, however, after emperical testing to lower mean ℓ_2-loss, we institute the following:

Date	Open	High	Low	Close	Adj Close	Volume
1/4/2010	7.6225	7.6607	7.5850	7.6432	6.4707	493729600
1/5/2010	7.6643	7.6996	7.6161	7.6564	6.4819	601904800
1/6/2010	7.6564	7.6868	7.5268	7.5346	6.3788	552160000

Table 1: AAPL Stock Price Dataset Head

- for RBF kernelization, we set $\gamma = 0.1$, and
- for the polynomial kernelization, we set $\gamma = 0.1$ and use a polynomial of degree $d = 2$.

3.4 Training and Testing Procedure

We select the 'Close' price of the stocks as the feature for predicting future prices and training the SVRs given its direct correlation to stock performance. The training set consists of data from January 1, 2010, to December 31, 2016. The testing data, i.e. where the model predicts the future prices of the stocks, spans from January 1st 2017 to January 14th, 2017. This splitting ensures that the model is trained on historical data and evaluated on more recent data, mimicking real-world scenarios.

We calulate the mean ℓ_2-loss of the approximated stock price to test the accuracy of the model. In the context of the stock prediction problem, the mean ℓ_2-loss is defined as

$$\bar{\mathcal{L}}(p, \hat{p}) = \frac{1}{n} \sum_{i=1}^{n} ||p_i - \hat{p}_i||_2^2 \tag{9}$$

where n is the number of days a stock price will be predicted, p_i is the predicted stock price using the SVR, and $\hat{p}_i$ is the actual stock price. The mean ℓ_2-loss allows us to determine an estimate for how accurate the SVR is for predicting stock price in a time interval, with lower values indicating a better approximation.

4 Results

The results for the predicted stock prices in the testing period January 1st, 2017 to January 14th, 2017 are summarized in Figure 5. The mean ℓ_2-loss for each kernelized SVR for their respective stocks are also summarized in Table 2. It is evident that the polynomial kernelization (despite using empirical measurements to lower its the mean ℓ_2-loss) results in a terrible approximation for the AAPL and COKE stocks, thus it is removed from all graphs in Figure 6 solely to display a more readable result.

	AAPL	COKE	GOOG	Mean
Linear	0.0265	2.611	0.267	0.968
RBF	0.0283	2.527	0.476	1.010
Polynomial	50.449	114.4104	2.409	55.756

Table 2: Mean ℓ_2-loss for Each Kernelized SVR for Each Stock

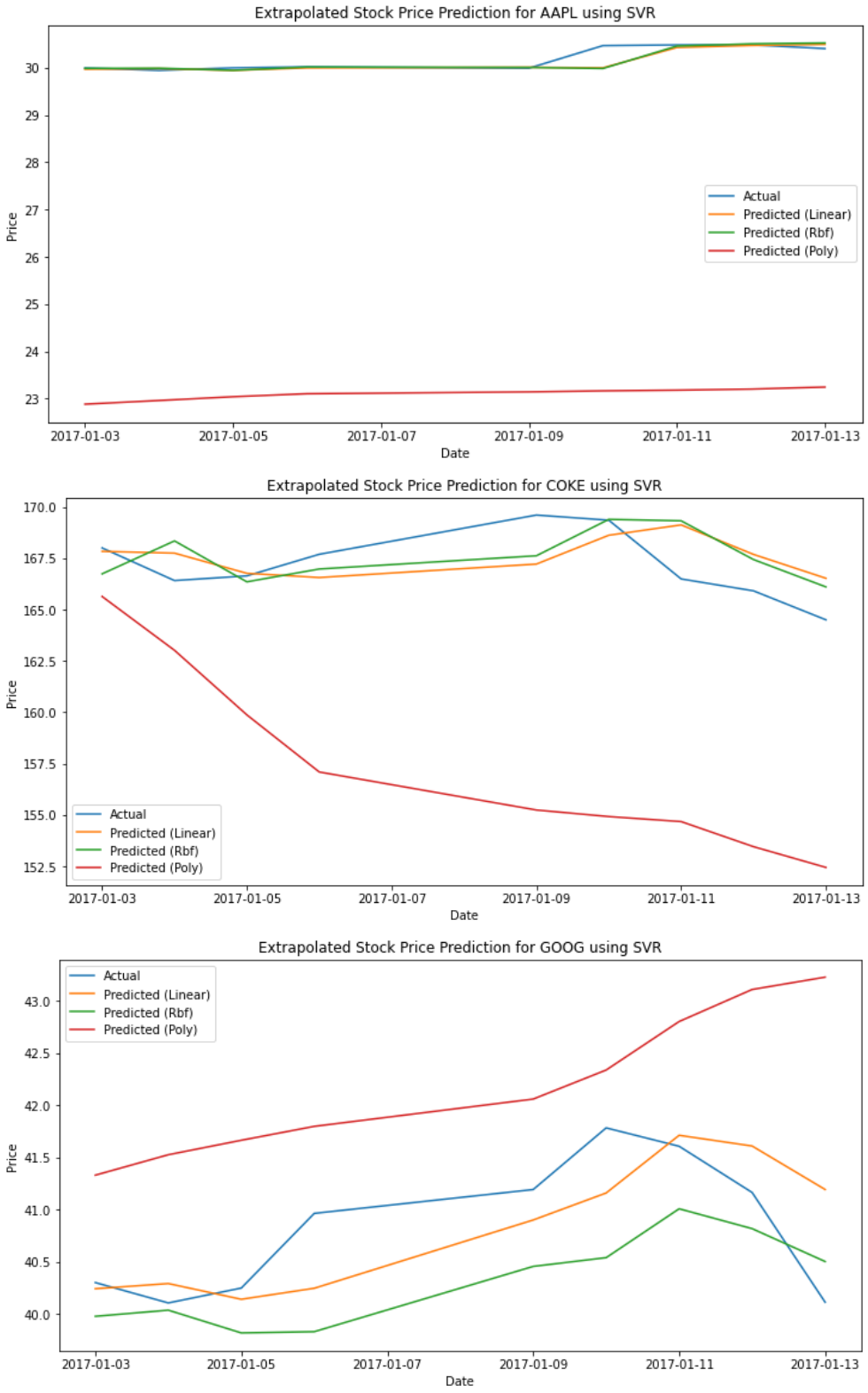

Figure 5: SVRs for Stock Price Prediction

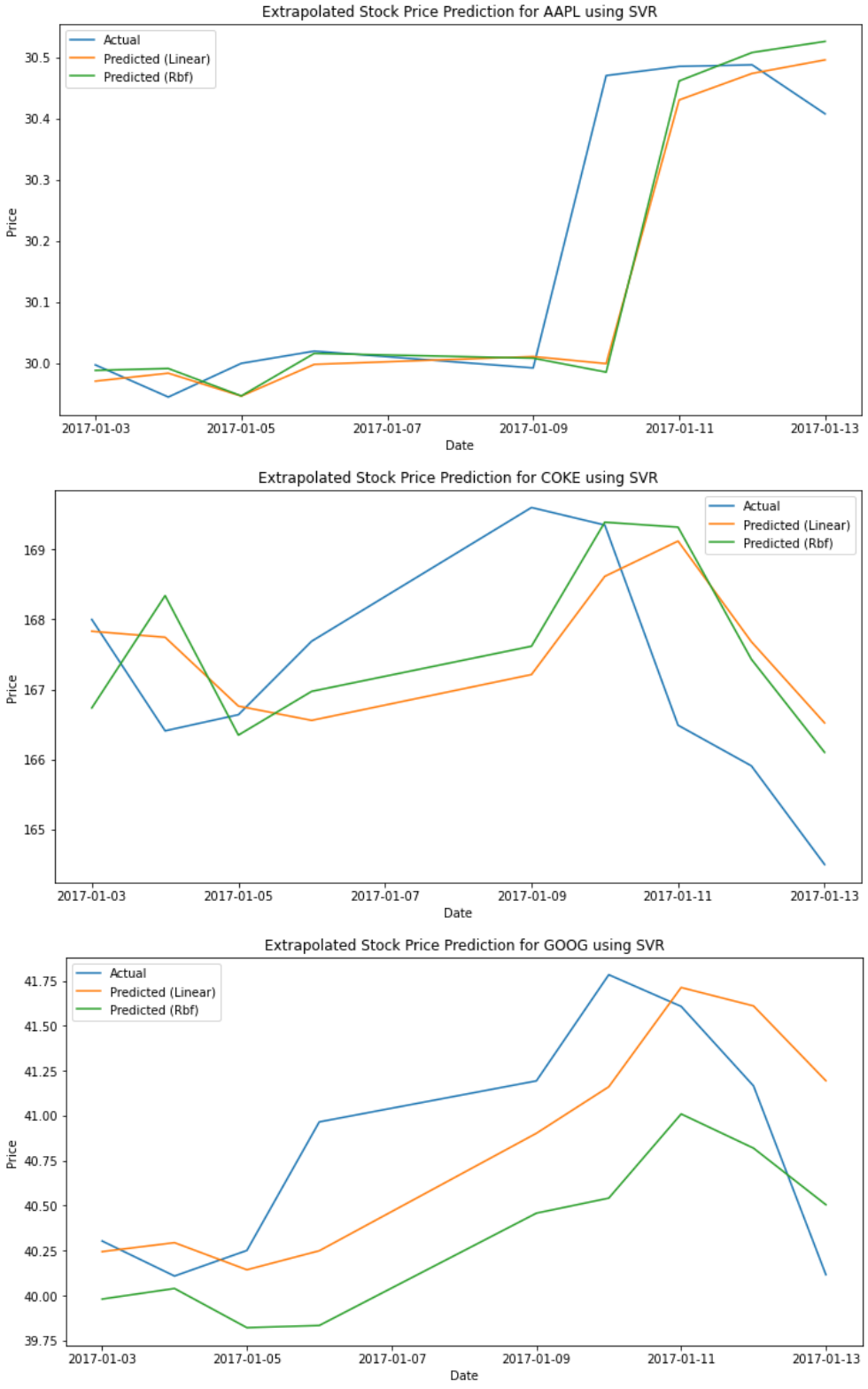

Figure 6: Improved SVRs for Stock Price Prediction

5 Deep Neural Networks: A Proven Alternative

5.1 A Brief Introduction to Deep Neural Networks

Although certain kernelizations of SVR provide promising results for the prediction of stock prices within a time interval, there exist many alternatives. One such alternative is a feed-forward deep neural network (DNN). The benefit of using a DNN is that its learning methods provide a robust approach to approximating real-valued functions [11]. There has also been extensive literature that supports their effectiveness in stock market prediction during the last decade [11]. Accordingly, we compare the effectiveness of the already-proven-effective DNN to that of the SVR for stock price forcasting.

5.2 Description of the Mathematics for DNNs

The focus of the description of the mathematics for the DNN will be on the actual architecture used to predict the stock price in this study and is derived from [12]. To understand DNNs, first consider the problem with an input space $\boldsymbol{\mathcal{X}} \in \mathbb{R}^n$, the closing price of a stock, and output space $\boldsymbol{\mathcal{Y}} \subseteq \mathbb{R}^p$, the predicted price of the stock for the next day. The goal of using a DNN is to determine a function $g : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{Y}}$ from a hypothesis set using training data $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ where $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ is the stock price and $\boldsymbol{y}^{(i)} \in \mathbb{R}^p$ is the corresponding day. In this case, the hypothesis is a collection of aritifical neural networks whose functions are created by the repeated and alternating application of a simple activation function.

The activation function used in each hidden layer of the DNN in the study is the Rectified Linear Unit (ReLU) function, or

$$\sigma(\boldsymbol{x}) = \max\{\boldsymbol{0}, \boldsymbol{x}\}.$$

The ReLu activation function introduces nonlinearity into the DNN, which is important because stock price prediction is a complex and nonlinear problem. ReLu is also computationally efficient, produces sparse activation which can avoid overfitting, and avoids the issue of vanishing gradient. All these components are essential for a regression problem like stock price prediction due to the high volume of training and testing data.

Let $\boldsymbol{x} \in \mathbb{R}^n$ be an input vector representing historical stock price data. The feedforward DNN used in this problem $g : \mathbb{R}^n \to \mathbb{R}^p$ is defined as a function that maps an input vector $\boldsymbol{x}$ from the input space $\mathbb{R}^n$ to an output vector $g(\boldsymbol{x})$ in the output space $\mathbb{R}^p$. The network consists of multiple layers:

- **Input Layer**: The initial activation $\boldsymbol{a}^{[0]}$ is set to the input vector $\boldsymbol{x}$, which is of dimension n.
- **Hidden Layers**: Each hidden layer $l = 1, 2, \ldots, L$ computes its activation $\boldsymbol{a}^{[l]}$ using the formula

  $$a^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$$

  where $\boldsymbol{W}^{[l]}$ is the weight matrix and $\boldsymbol{b}^{[l]}$ is the bias vector associated with layer l. The resulting activation $\boldsymbol{a}^{[l]}$ is in the space $\mathbb{R}^{n_{[l]}}$, where $n_{[l]}$ is the number of neurons in layer l.
- **Output Layer**: The final output $g(\boldsymbol{x}) = \boldsymbol{a}^{[L]}$ is obtained from the activation of the last layer L, which produces a vector in the output space $\mathbb{R}^p$, representing the predicted stock price.

The neural network parameters $\boldsymbol{W}^{[l]}$ and $\boldsymbol{b}^{[l]}$ are learned during training using historical stock data. The model is trained to minimize the prediction error (ℓ_2-loss) between the predicted stock prices and the actual prices. Once trained, the neural network can be used to make predictions on new or extrapolated data, providing insights into future stock price trends

The architecture of the DNN for stock price prediction in this study is comprised of five total layers ($L = 5$), including an input layer which takes in the closing price of a stock, three hidden layers, and an output layer which predicts the new price of the stock the following day. Each hidden layer contains 32 nodes ($n_{[l]} = 32$) and applies a series of weighting and biasing through a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$. The weight matrices and bias vectors are learned by training the model to capture the complex patterns and relationships between the historical stock prices and the next day's stock price. They also allow the model to learn the overall trends in the data, affecting the activation of neurons within the model and the final prediction.

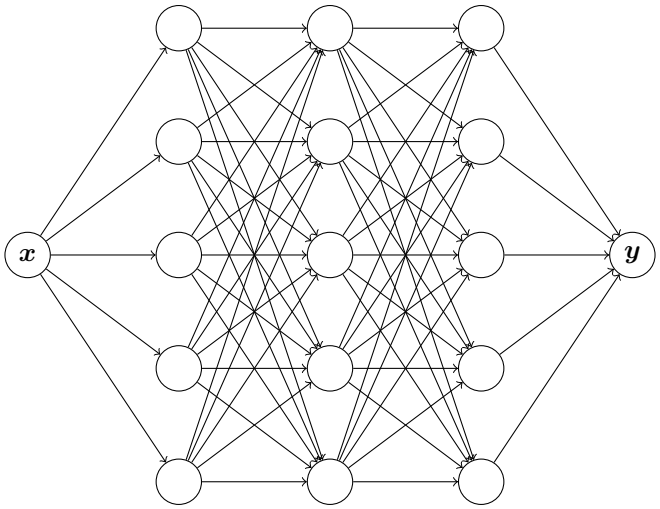

Figure 7: Scaled-Down NN

The ADAM optimizer, short for Adaptive Moment Estimation, is a powerful optimization algorithm widely employed in training deep neural networks, including those used for stock price prediction. ADAM computes individual learning rates for each parameter based on estimates of first-order (mean) and second-order (uncentered variance) moments of gradients. This adaptive learning rate mechanism is particularly effective in navigating complex loss landscapes and handling large, high-dimensional datasets common in financial applications. ADAM's integration of momentum optimization and bias-correction enhances convergence speed and stability during training. In the context of predicting stock prices with DNNs, ADAM optimizes the model's parameters to minimize the ℓ_2-loss between predicted and actual prices. A scaled-down model of the DNN in this investigation is displayed in Figure 7.

5.3 Results of the DNN

The results for the predicted stock prices in the testing period January 1st, 2017 to January 14th, 2017 are summarized in Figure 8. The mean ℓ_2-loss for each stock price is summarized in Table 3. Based on the results, it is evident that the DNN, as expected, provides a robust approximation for a variety of different stocks, with accuracy similar to that of the linearly kernelized SVR.

AAPL	COKE	GOOG
0.0282	1.707	0.131

Table 3: Mean ℓ_2-loss for Each Stock Price Using DNN

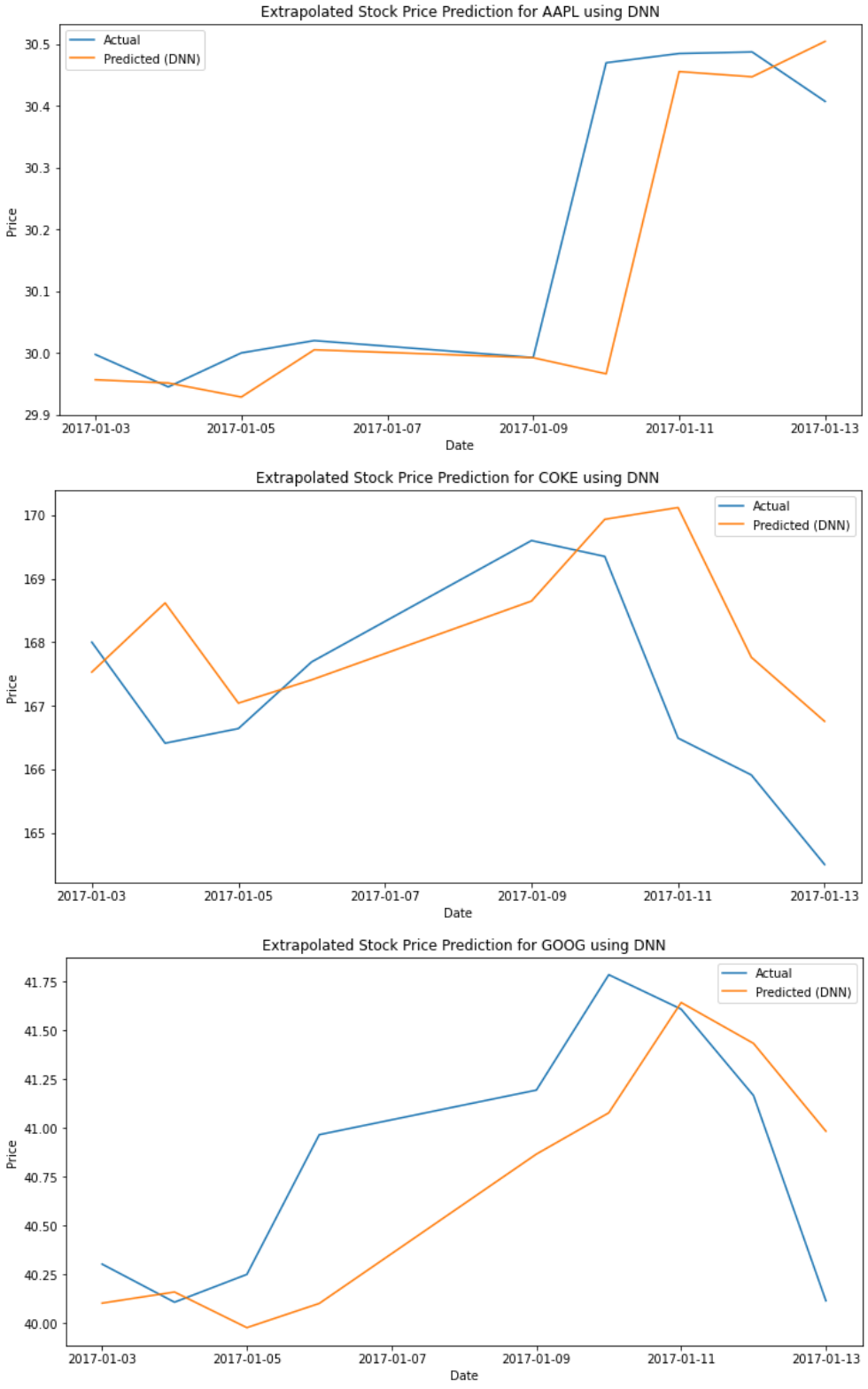

Figure 8: DNN for Stock Prediction

6 Conclusions

The development of predictive models for the stock market is an imperitave task for the maximization of returns on investment and profit. By developing computationally intensive methods, investors can harness historical stock price data to generate models to forecast future stock prices for a short window. Of the computational and machine learning methods available, this study harnesses the power of SVR and measures its performance on the AAPL, COKE, and GOOG stock prices, and compares it to the proven method of DNN regression. To determine the accuracy of the forecasted stock prices, we compute the mean ℓ_2-loss for each kernel of the SVR, namely the linear, RBF, and polynomial kernels.

The study demonstrates that to minimze the ℓ_2-loss of the predicted stock price, one should use the linear kernel. Although the performance of the linear kernel was very comparable to that of the RBF kernel, it performed slightly better in the small sample size produced in the study. The study also demonstraes that the polynomial kernel of degree two performs considerably worse than other kernels for SVR. For the successful kernelizations, their accuracy was comparable to that created by the DNN.

An important contribution by this paper and these findings is the empirical validation of support vector regression (SVR) as a viable method for forecasting future stock prices within a short window. By harnessing the power of SVR and evaluating its performance on real-world stock price data from companies like AAPL, COKE, and GOOG, this study provides valuable insights into the effectiveness of different kernel functions in SVR. The paper also encourages researchers and practitioners to carefully select kernel functions based on their specific dataset characteristics and prediction objectives.

While this research offers valuable insights, it is important to recognize its limitations. The study's narrow focus on just three stocks (AAPL, COKE, and GOOG) may limit the generalizability of its findings. Including a broader range of stocks from diverse sectors and market capitalizations could offer a more comprehensive understanding of SVR's performance across various market conditions. Moreover, the training period from January 1, 2010, to December 31, 2016 might impact the predictive accuracy of the SVR models. Extending the training period or exploring different time intervals could enhance the models' performance and robustness. By relying solely on empirical data to determine kernel function parameters (e.g. and γ for the RBF kernel, degree for the polynomial kernel), it is possible we overlook the full complexity of the underlying data distribution. Utilizing more sophisticated optimization techniques or conducting sensitivity analyses could improve parameter selection. The presence of missing data (although rare in the datset) poses another challenge, potentially introducing biases or inaccuracies in the model training and evaluation. Effective handling of missing data, through techniques like imputation or data preprocessing, is crucial for ensuring result integrity and reliability. The exclusive focus on mean ℓ_2-loss as the evaluation metric may neglect other important aspects of model performance, such as volatility prediction, outlier detection, or interpretability. Exploring alternative evaluation metrics or employing ensemble methods could offer a more comprehensive assessment of SVR's effectiveness in stock price prediction.

Further studies may consequently address the many limitation of this study. Some examples include expanding the scope of analysis to include a more diverse set of stock across various sectors; developing a sophisticating approach to kernel parameter tuning; exploring alternative evaluation metrics to calculate the accuracy of the model; or incorporating more features in the model training to create a more robust training framework. A crucial investigation could be carried out into other, perhaps better, machine learning techniques for stock price prediction.

References

[1] Alan Fan and Marimuthu Palaniswami. Stock selection using support vector machines. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1793–1798. IEEE, 2001.

[2] Sui Xue-shen, Qi Zhong-ying, Yu Da-Ren, Hu Qing-hua, and Zhao Hui. A novel feature selection approach using classification complexity for svm of stock market trend prediction. In *2007 International conference on management science and engineering*, pages 1654–1659. IEEE, 2007.

[3] Veronica Piccialli and Marco Sciandrone. Nonlinear optimization and support vector machines. *Annals of Operations Research*, 314(1):15–47, 2022.

[4] Wei Huang, Yoshiteru Nakamori, and Shou-Yang Wang. Forecasting stock market movement direction with support vector machine. *Computers & operations research*, 32(10):2513–2522, 2005.

[5] Nick Dexter. Machine learning iii: Binary classification and support vector machines. *Florida State University*, 2022.

[6] Mathematical introduction to svm and kernel methods.

[7] Dustin Boswell. Introduction to support vector machines. *Departement of Computer Science and Engineering University of California San Diego*, 11, 2002.

[8] Radial basis function (rbf) kernel: The go-to kernel.

[9] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(4), 2010.

[10] Mariette Awad, Rahul Khanna, Mariette Awad, and Rahul Khanna. Support vector regression. *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pages 67–80, 2015.

[11] Suraiya Jabin. Stock market prediction using feed-forward artificial neural network. *International Journal of Computer Applications*, 99(9):4–8, 2014.

[12] Nick Dexter. Deep learning i: Introduction and basic concepts. *Florida State University*, 2022.