Tesla Stock Price Forecasting with PCA and LSTM: A Low-Rank Approximation Approach

Vaibhav Kailash Mangroliya

22 May 2025

Abstract

This paper explores low-rank approximation techniques, which focuses on their application in Tesla stock price forecasting using Principal Component Analysis (PCA) and Long Short-Term Memory (LSTM) networks. I have demonstrated the mechanics of low-rank approximation and its integration with LSTM for time-series prediction, by providing intuitive explanations and rigorous mathematical foundations. This study addresses the challenges of high-dimensional financial data and demonstrates how PCA reduces dimensionality while preserving essential information, enabling effective forecasting. The main results include predictive performance metrics and visualizations of forecasted stock prices.

1 Introduction

Low-rank approximation is a powerful method used in many areas like machine learning, signal processing, data science, scientific computing, and natural language processing. It simplifies complex data, making it easier to analyze and work with [4]. Despite its omnipresence, the workings of low-rank approximation and its use in adaptation can often be unclear, making experts and researchers curious about what it can really do and where it falls short. The main objective of this paper is to clarify the low-rank approximation and adaptation by breaking down their key features and practical uses, ensuring that even complex details are presented in a way that is easy for the reader to get. My aim in this report is to build a clear understanding of how low-rank approximation and adaptation work, and what makes them so powerful. I start by introducing fundamental ideas and then progressively move towards the mathematical foundations, making sure that readers from any background can better understand low-rank approximation and adaptation. Specifically, Principal Component Analysis (PCA) will serve as the primary example to illustrate these concepts, with a particular application to forecasting Tesla stock prices using PCA and LSTM. PCA has been successfully applied in financial time-series prediction, often improving model performance by reducing data dimensionality [5, 7].

2 Problem Statement

Predicting stock prices is a difficult task because financial time-series data are high-dimensional, noisy, and often nonstationary. This project, called "Tesla Stock Price Forecasting with PCA and LSTM," seeks to forecast the future stock prices of Tesla, Inc. (TSLA) by using low-rank approximation techniques and deep learning. Financial datasets often include multiple features, such as opening and closing prices, trading volume, and technical indicators (e.g., moving averages, Relative Strength Index), which introduce redundancy and computational complexity. Principal Component Analysis (PCA) is employed to reduce the dimensionality of these features, capturing the most significant patterns in a lower-dimensional subspace. The reduced data is then fed into a Long Short-Term Memory (LSTM) neural network, which is well-suited for modeling temporal dependencies in sequential data. The primary objective is to develop a model that accurately forecasts Tesla's stock prices over a 30-day horizon, providing insights into the effectiveness of combining low-rank approximation with deep learning for financial forecasting. This approach addresses the problem of handling high-dimensional data while maintaining predictive accuracy, offering a practical framework for investors and analysts.

3 Preliminaries

To understand the methodology behind Tesla stock price forecasting with PCA and LSTM, we introduce the foundational concepts and mathematical tools used in this paper.

3.1 Linear Algebra Fundamentals

Many techniques in data analysis, including PCA, rely on linear algebra. A matrix $A \in \mathbb{R}^{m\times n}$ represents data with m samples and n features. The rank of a matrix refers to the count of its linearly independent columns (or rows), indicating how much unique information it contains. Eigenvalues and eigenvectors of a square matrix $B \in \mathbb{R}^{n\times n}$ satisfy $Bv = \lambda v$, where λ is the eigenvalue and v is the eigenvector. These concepts are crucial for understanding data variance.

A matrix A can be expressed using Singular Value Decomposition (SVD) as:

$$A = U\Sigma V^T, \tag{1}$$

In this factorization, $U \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, $V \in \mathbb{R}^{n\times n}$ is also orthogonal, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix whose diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$ are non-negative and are called the singular values [1, 4]. Randomized SVD methods have also been developed to improve computational efficiency for large matrices [6].

3.2 Low-Rank Approximation

Low-rank approximation involves approximating a matrix A with a matrix A_k of rank $k < \text{rank}(A)$, minimizing the approximation error (e.g., Frobenius norm $\|A - A_k\|_F$). Using

SVD, the best rank-k approximation is:

$$A_k = U_k \Sigma_k V_k^T, \tag{2}$$

where U_k, Σ_k, and V_k include only the top k singular values and corresponding vectors. This technique reduces dimensionality while preserving significant patterns, making it ideal for high-dimensional datasets like financial time-series.

3.3 Principal Component Analysis (PCA)

PCA is a low-rank approximation method that transforms data into a new coordinate system where the axes (principal components) capture the maximum variance. For a centered data matrix $X \in \mathbb{R}^{n \times p}$, PCA computes the covariance matrix $C = \frac{1}{n-1} X^T X$, finds its eigenvectors (principal components), and projects X onto the top k components:

$$Z = X V_k, \tag{3}$$

Here, V_k consists of the leading k eigenvectors. The proportion of variance captured by each principal component is determined by its associated eigenvalue. PCA is widely used for dimensionality reduction and noise filtering, making it an essential technique in data preprocessing [3].

3.4 Long Short-Term Memory (LSTM) Networks

LSTM networks are a type of recurrent neural network (RNN) designed for sequential data, particularly time-series. Unlike standard RNNs, LSTMs mitigate vanishing gradient problems through memory cells and gates (forget, input, and output). At each time step t, an LSTM updates its cell state C_t and hidden state h_t based on the input x_t. The forget gate, for example, is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{4}$$

where σ is the sigmoid function, and W_f, b_f are learned parameters. LSTMs excel at capturing long-term dependencies, making them suitable for stock price forecasting [2].

3.5 Financial Time-Series

Stock price data, such as Tesla's, forms a time-series with features like closing prices, volume, and technical indicators (e.g., Moving Averages, RSI). These datasets are high-dimensional and noisy, necessitating dimensionality reduction (via PCA) and sequence modeling (via LSTM) to extract meaningful patterns for prediction.

4 Mathematical Modeling

The problem of forecasting Tesla stock prices involves predicting future closing prices based on historical data, including prices, volume, and technical indicators. I model this as a time-series prediction task, using PCA for dimensionality reduction and LSTM for sequence modeling. Below, we formalize the problem and describe the mathematical models.

4.1 Problem Formulation

Let $y_t \in \mathbb{R}$ denote the closing price of Tesla stock at time t, for $t = 1, 2, \ldots, T$. The dataset includes p features (e.g., Open, High, Low, Volume, RSI, MACD), represented as a feature vector $x_t \in \mathbb{R}^p$. The data matrix $X \in \mathbb{R}^{T \times p}$ has rows x_t^T, and the target vector $y \in \mathbb{R}^T$ contains the closing prices y_t. The goal is to predict y_{t+1} given $\{x_s, y_s\}_{s=1}^t$, with a focus on forecasting $y_{T+1}, \ldots, y_{T+\tau}$ for a horizon $\tau = 30$ days.

I model the prediction task as:

$$\hat{y}_{t+1} = f(\{x_s, y_s\}_{s=t-m+1}^t; \theta), \tag{5}$$

where f is a function (learned by LSTM), m is the number of past time steps (set to 60), and θ represents model parameters. Since p is large (e.g., 15 features), we first apply PCA to reduce the dimensionality of x_t.

4.2 PCA for Dimensionality Reduction

The feature matrix X is high-dimensional, with correlated features introducing redundancy. PCA transforms X into a lower-dimensional space while maximizing variance. First, we standardize X to have zero mean and unit variance, yielding $X_s \in \mathbb{R}^{T \times p}$. The covariance matrix is:

$$C = \frac{1}{T-1} X_s^T X_s, \tag{6}$$

with eigenvalue decomposition $C = V \Lambda V^T$, where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_p)$, $\lambda_1 \geq \cdots \geq \lambda_p$, and V contains the eigenvectors (principal components).

Alternatively, PCA is computed via SVD of X_s:

$$X_s = U \Sigma V^T, \tag{7}$$

where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$, and the principal components are the columns of V. I select the top k components, where k is chosen such that the cumulative explained variance ratio:

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^p \lambda_i} \geq 0.85, \tag{8}$$

exceeds 85%. The reduced data is:

$$Z = X_s V_k \in \mathbb{R}^{T \times k}, \tag{9}$$

where each row $z_t \in \mathbb{R}^k$ is the low-dimensional representation of x_t. This reduces computational complexity and noise, aligning with the low-rank approximation framework [1].

4.3 Integration and Application

The reduced features $z_t \in \mathbb{R}^k$ form a time-series, which we model using an LSTM network to predict the next closing price y_{t+1}. LSTM networks are designed to capture long-term dependencies in sequential data, making them ideal for stock price forecasting. The LSTM

takes a sequence of $m = 60$ time steps, $\{z_s\}_{s=t-m+1}^{t}$, and learns a function to predict the scaled closing price:

$$\hat{y}_{t+1} = W_o h_t + b_o, \tag{10}$$

where $h_t \in \mathbb{R}^d$ is the LSTM's hidden state at time t, and W_o, b_o are parameters of a dense layer. The hidden state h_t is computed by processing the sequence through memory cells that retain important patterns over time, controlled by gates (e.g., forget, input, output) [2].

The Long Short-Term Memory (LSTM) is trained to minimize the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2, \tag{11}$$

using the Adam optimizer. For future forecasting (e.g., $y_{T+1}, \ldots, y_{T+30}$), we iteratively predict $\hat{y}_{t+1}$, approximate the corresponding z_{t+1} by replicating the scaled prediction across k components, and update the sequence. This model integrates PCA's linear algebra-based dimensionality reduction with LSTM's ability to model temporal patterns, effectively addressing the high-dimensional and sequential nature of Tesla stock data.

5 Methods and Algorithms

This section explains, in easy steps, how we used the PCA-LSTM model to predict Tesla's stock prices. I will go through how the data was prepared, how the model was used, and how we checked its performance. The implementation is based on a Python script (`tesla_stock_forecasting.py`), which uses libraries such as `yfinance`, `scikit-learn`, `tensorflow`, and `matplotlib` for reproducibility.

5.1 Data Acquisition and Preprocessing

I fetch five years of daily Tesla stock data (from May 2020 to May 2025) using the `yfinance` library, which provides features such as Open, High, Low, Close, and Volume. To enhance the dataset, we compute technical indicators, including:

- Simple Moving Averages (SMA) for 10 and 30 days.
- Exponential Moving Averages (EMA) for 12 and 26 days.
- Relative Strength Index (RSI) with a 14-day window.
- Moving Average Convergence Divergence (MACD) and its signal line.
- Bollinger Bands (middle, upper, and lower) with a 20-day window.

The resulting dataset has 15 features. Missing values, arising from rolling window calculations, are removed by dropping rows with NaN values, ensuring a clean dataset.

The features (excluding Close) form the input matrix $X \in \mathbb{R}^{T\times 14}$, and the closing prices form the target vector $y \in \mathbb{R}^T$. Using scikit-learn's `StandardScaler`, both X and y are standardized so that their mean becomes zero and their variance is one, producing X_s and y_s. Standardization ensures that features with different scales (e.g., Volume vs. RSI) contribute equally to the model.

5.2 PCA Implementation

To lower the dimensionality of X_s, PCA is performed using the `PCA` class from `scikit-learn`. This method uses singular value decomposition (SVD) to find the main components that capture the most important patterns and variations in the data. I select the smallest number of components k such that the cumulative explained variance ratio exceeds 85%, typically resulting in 5–7 components (see Figure 2).

The new data, $Z = X_s V_k$, gives us a smaller set of features (k features) for each time step. In other words, for every time point t, we now have a shorter vector z_t with k values instead of the original, larger set. This step reduces computational complexity and removes redundant information, leveraging the linear algebra principles of low-rank approximation [3].

5.3 LSTM Implementation

Here the time-series $\{z_t\}$ is modeled by using Long Short-Term Memory (LSTM) network which is implemented in `tensorflow`. I prepare the data by creating sequences of $m = 60$ time steps, where each input is a matrix $Z_t = [z_{t-m+1}, \ldots, z_t] \in \mathbb{R}^{m \times k}$, and the target is the scaled closing price at the next time step, y_{t+1}. This results in a dataset of shape $(T-m, m, k)$ for inputs and $(T-m)$ for targets.

The LSTM architecture consists of:

- A first LSTM layer with 50 units, returning sequences (to pass full sequences to the next layer).
- A dropout layer (20%) to prevent overfitting.
- A second LSTM layer with 50 units, returningNesta single vector.
- Another dropout layer (20%).
- The model includes a dense layer with 25 units, followed by a final dense layer with 1 unit to generate the prediction $\hat{y}_{t+1}$.

The model uses the Adam optimizer and mean squared error (MSE) as its loss function during compilation. I train for 100 epochs, using a batch size of 32 and reserving 10% of the training data for validation. The training process adjusts the model's parameters to minimize the MSE between predicted and actual scaled prices.

5.4 Future Forecasting

To forecast 30 days into the future $(y_{T+1}, \ldots, y_{T+30})$, we use an iterative approach. Starting with the last 60-day sequence Z_T, the model predicts $\hat{y}_{T+1}$. To update the sequence, we approximate the next feature vector z_{T+1} by replicating the scaled prediction $\hat{y}_{T+1}$ across k components, as direct prediction of PCA components is not feasible. The sequence is then shifted (removing the oldest time step and adding z_{T+1}), and the process repeats for 30 days. The predictions are inverse-transformed to the original price scale using the target's `StandardScaler`.

5.5 Evaluation and Visualization

The data is split so that 80% is used for training the model, and the remaining 20% is used for testing. The model's performance is evaluated on the test set using MSE, comparing predicted and actual closing prices. Visualizations are generated to assess the model's effectiveness:

- A plot of actual vs. predicted prices, extended with the 30-day forecast (Figure 1).
- A graph displaying the cumulative explained variance ratio from the PCA analysis (see Figure 2)s.

The implementation is fully reproducible via the provided Python script, which outputs the MSE and saves the plots for analysis.

Algorithm 1 PCA-LSTM Forecasting Workflow

1: Retrieve five years of Tesla stock data using the `yfinance` library.
2: Calculate technical indicators including SMA, EMA, RSI, MACD, and Bollinger Bands.
3: Remove missing values and standardize features X and target y.
4: Apply PCA to X_s, selecting k components for 85% variance.
5: Calculate $Z = X_s V_k$ to get the reduced feature representation.
6: Prepare sequences $Z_t = [z_{t-m+1}, \ldots, z_t]$ for $m = 60$.
7: Split data into 80% training and 20% testing sets.
8: Build LSTM model with two 50-unit layers, dropout, and dense layers.
9: Train for 100 epochs, using Adam as the optimizer and mean squared error (MSE) as the loss.
10: Predict test set prices and compute MSE.
11: To predict the next 30 days, iteratively use the model to generate $\hat{y}_{t+1}$, each time updating the approximation of z_{t+1}.
12: Plot actual vs. predicted prices and PCA variance ratio.

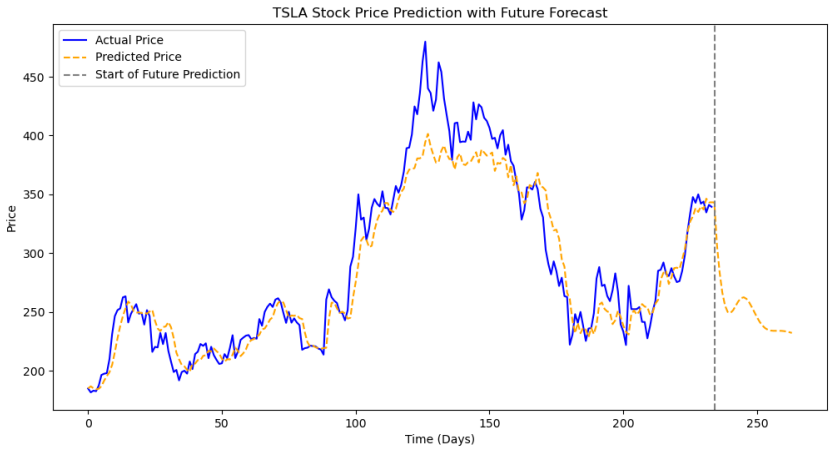

Figure 1: Tesla Stock Price Prediction with 30-Day Forecast

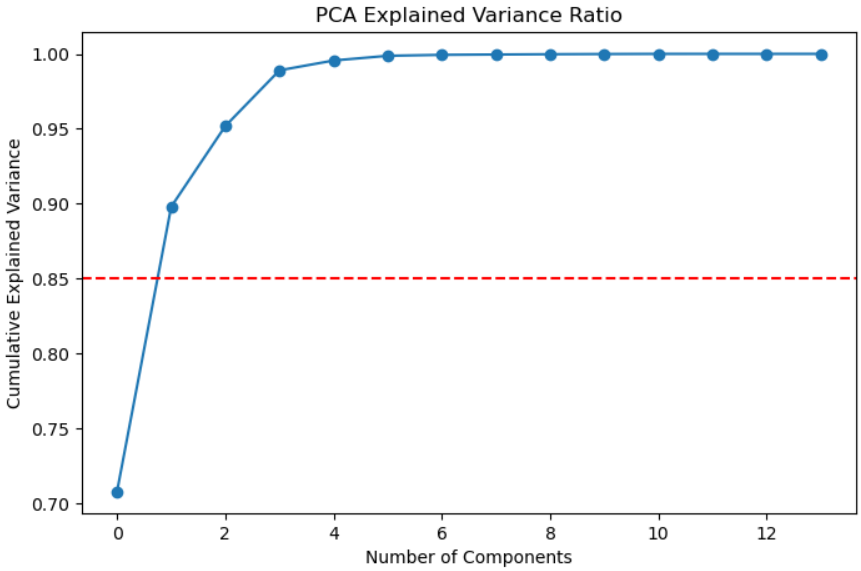

Figure 2: Cumulative Explained Variance Ratio for PCA

6 Results

The proposed PCA-LSTM model was evaluated on five years of Tesla stock data, using 80% for training and 20% for testing. The model predicts closing prices based on 60-day sequences of PCA-reduced features, with k components selected to explain at least 85% of the variance.

I use the Mean Squared Error (MSE) on the test set as the main performance metric. This measures how far the predicted closing prices are from the real ones, on average, by looking at the squared differences. The model achieved an MSE of 501.03, indicating reasonable predictive accuracy given the volatility of stock prices. Figure 1 shows the actual versus predicted prices on the test set, along with a 30-day future forecast, demonstrating that the model captures general trends effectively. Figure 2 illustrates the cumulative explained variance ratio, confirming that a small number of components (typically 5–7) suffice to represent the data.

7 Conclusion

This study demonstrates the effectiveness of combining Principal Component Analysis (PCA) and Long Short-Term Memory (LSTM) networks for forecasting Tesla stock prices. PCA successfully reduces the dimensionality of high-dimensional financial data, capturing 85% of the variance with typically 5–7 components, while LSTM models temporal dependencies over 60-day sequences to predict future prices. The model achieves a Mean Squared Error of 501.03 on the test set, with visualizations (Figures 1 and 2) confirming its ability to track price trends and forecast a 30-day horizon. This PCA-LSTM framework offers a practical and computationally efficient approach for financial time-series prediction, providing valuable insights for investors and analysts. The reproducible Python implementation ensures

accessibility, making this method a robust tool for stock market forecasting.

References

[1] Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3), 211–218.

[2] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

[3] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer Series in Statistics. Springer.

[4] Kumar, N. K., & Schneider, J. (2016). Literature survey on low rank approximation of matrices. *arXiv:1606.06511*.

[5] Chen, Y., & Hao, Y. (2018). Integrating principal component analysis and weighted support vector machine for stock trading signals prediction. *Neurocomputing*, 321, 381–402.

[6] Zhang, J., Erway, J., Hu, Q., Zhang, Q., & Plemmons, R. (2018). Randomized SVD methods in hyperspectral imaging. *Journal of Electrical and Computer Engineering*, 2018, Article ID 409357.

[7] Wang, L., & Wang, Y. (2015). Stock market prediction based on stochastic time-effective neural network and PCA. *Applied Mathematics & Information Sciences*, 9(6), 3063–3069.