HW4 110034025

1. In this experiment, we evaluated three different combinations of window size and step, as shown in the following table 1. When using a small window size 5 with a step of 1, the model achieved the best performance with a low test MSE of 2.7605. However, when the window size increased to 10 and 20, and the step sizes were enlarged to 5 and 10, the test MSE rose significantly, reaching 157.2513 and 492.1293, respectively. This suggests that a smaller window with a finer step captures short-term patterns better in stock price prediction, whereas larger windows may introduce noise and make learning more difficult.

Table 1 Different combinations of window size and step

| Configuration | Train Loss | Val Loss | Test MSE |
|---|---|---|---|
| Window: 5, Step: 1 | 1.6517 | 2.1506 | **2.7605** |
| Window: 10, Step: 5 | 104.3581 | 134.5102 | 157.2513 |
| Window: 20, Step: 10 | 265.8211 | 132.0193 | 492.1293 |

2.

(i) Including "Volume" as an additional feature worsened the model's performance significantly while window and step is set to (5,1), as shown in the following table 2. Without "Volume," the model achieved a test MSE of 23.3947, while including "Volume" increased the MSE to 957.5522, showing a degradation of approximately 3993.03%. This suggests that "Volume" may introduce noise or irrelevant variability for short-term stock price prediction in this case, making the learning process harder instead of helping it.

Table 2 Impact of incorporating 'Volume' as an additional feature

| Configuration | Train Loss | Val Loss | Test MSE |
|---|---|---|---|
| Without Volume | 19.4855 | 22.3057 | **23.3947** |
| With Volume | 943.4546 | 880.3029 | 957.5522 |

(ii) To find the best feature combination, we tested various subsets randomly of the input features while window and step is set to (5,1), as shown in the following table 3. The combination of ['High', 'Low'] yielded the lowest test MSE of 24.0754, outperforming combinations that included "Volume" or "OpenInt." Combinations like ['Open', 'High', 'Low', 'Close'] and ['Open', 'Close'] performed decently with a close result with the best one. These results indicate that the core price-related features ("Open", "High", "Low", "Close") contain the most relevant information for predicting the next day's "High" price, while additional features like "Volume" or "OpenInt" add unnecessary noise rather than providing useful predictive signals.

Table 3 Different combination of input features

| Features | Train Loss | Val Loss | Test MSE |
|---|---|---|---|
| Open, Close | 22.2579 | 27.2465 | 26.0897 |
| High, Low | 21.5820 | 17.0010 | **24.0754** |
| Open, High, Low, Close | 24.9397 | 11.6703 | 25.6314 |
| Open, High, Low, Close, Volume | 900.3758 | 1057.3024 | 957.6337 |
| Open, Close, Volume | 938.6877 | 917.4312 | 957.5554 |
| High, Volume | 929.4512 | 928.1915 | 957.6382 |
| Close, Volume, OpenInt | 925.8700 | 944.4177 | 957.8044 |

3. The experimental results clearly show that normalization significantly improves model performance in Lab 4, as shown in the following table 4. We use features including ['Open', 'High', 'Low', 'Close', 'Volume'] to test out the results, while window and step is set to (5,1). Without normalization, the model achieved a high test MSE of 957.6982, whereas with normalization, MSEs dropped dramatically. Specifically, robust normalization achieved the best performance, reaching an MSE of 2.0557 and RMSE of 1.4338. MinMax and standard normalization also performed much better than no normalization but were slightly worse than robust. Therefore, we can see that normalization helps the model converge better.

Table 4 Performance of the model with normalization

| Method | Train Loss | Val Loss | Test MSE | Test RMSE |
|---|---|---|---|---|
| None | 938.4832 | 897.6890 | 957.6982 | 30.9467 |
| MinMax | 0.0001 | 0.0000 | 2.3317 | 1.5270 |
| Standard | 0.0018 | 0.0027 | 2.1232 | 1.4571 |
| Robust | 0.0122 | 0.0150 | **2.0557** | **1.4338** |

4. In time-series forecasting, the window size typically represents how much past information the model sees, while the step size controls how much the window shifts. Window size should not be less than step size, because this risks skipping important temporal patterns. Instead, usually window size ≥ step size to maintain sequential continuity (Time Series Forecasting as Supervised Learning[1]).

5. One method for time-series data augmentation is jittering, which adds random noise to the time-series values to simulate natural variations and improve model robustness. For example, Gaussian noise can be applied to numerical time-series data points within a specified standard deviation. This helps models generalize

[1] Time Series Forecasting as Supervised Learning, https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

better by exposing them to varied data patterns. According to Wen et al. (2020), jittering is effective for time-series classification tasks, enhancing model performance on noisy real-world data (Data augmentation for time series: Traditional vs generative models on capacitive proximity time series[2]).

6.

(i) In convolutional networks (e.g., CNNs), the model usually expects a fixed-size input window during inference, matching the training window size. Since convolutions are local and translation-invariant, sliding the window across the time series (with or without overlap) is a common practice for prediction. Fixed window size is critical to maintain the learned receptive fields.

(ii) Recurrent models like LSTM or GRU can, in theory, handle variable-length sequences during inference. However, if trained with a fixed window size, it's generally best to provide the same size during inference to maintain consistency in learned temporal dependencies. Otherwise, padding or truncation may be needed.

(iii) Transformers typically require fixed-length input during inference because positional encoding depends on the sequence length seen during training. A mismatch could lead to degraded performance. Like CNNs, one can slide a fixed-size window over the sequence for prediction.

[2] Wen et al. (2020), Data augmentation for time series: Traditional vs generative models on capacitive proximity time series