**LSTM:**

1. **Increased Model Complexity:**
   * **In the original code, a simple LSTM model with one LSTM layer containing 50 units was used.**
   * **In the improved code, the model complexity was increased by adding an additional LSTM layer with 100 units and specifying return\_sequences=True in the first LSTM layer. This configuration allows the output sequence of the first LSTM layer to be used as input for the second LSTM layer, effectively stacking two LSTM layers.**
   * **Increasing the number of LSTM units and adding layers allows the model to capture more complex patterns and relationships in the data, potentially leading to better prediction performance.**
2. **Adjusted Training Parameters:**
   * **The number of epochs was increased from 10 to 20 (epochs=20), allowing the model to undergo more training iterations.**
   * **The batch size was adjusted from 1 to 32 (batch\_size=32), determining the number of samples processed before updating the model's weights.**
   * **These adjustments provide the model with more opportunities to learn from the data and refine its predictions.**
3. **Data Normalization:**
   * **Min-Max scaling was applied to normalize the closing price data to a range between 0 and 1.**
   * **The MinMaxScaler from scikit-learn was used to perform this normalization (scaler.fit\_transform(data)).**
   * **Normalizing the data helps stabilize the training process and ensures that all features contribute equally to the model's learning process, potentially improving convergence and prediction accuracy.**
4. **Feature Engineering:**
   * **Feature engineering involves selecting and transforming relevant input features to improve model performance.**
   * **In this code, only the closing price was used as input for prediction.**
   * **Although not implemented in the provided code, incorporating additional features such as volume, high, low, etc., could potentially enhance the model's predictive capabilities by providing more information for learning.**
5. **Regularization:**
   * **Regularization techniques such as dropout layers were not explicitly implemented in the provided code but can be beneficial for preventing overfitting.**
   * **Dropout layers randomly deactivate a fraction of neurons during training, forcing the model to learn more robust and generalized representations of the data.**
6. **Evaluation and Analysis:**
   * **The performance of the model was evaluated using the root mean squared error (RMSE) between the predicted and actual closing prices.**
   * **RMSE was calculated and normalized by dividing by the range of the target variable, providing a measure of prediction accuracy relative to the scale of the data.**
   * **By analyzing the RMSE, insights into the model's predictive accuracy can be gained, guiding further improvements or adjustments.**

**KNN**

1. **Feature Scaling**:
   * **We use Min-Max scaling (MinMaxScaler) to normalize the closing price data between 0 and 1 (data\_scaled). Normalization ensures that all features are on a similar scale, which can aid in the convergence of the KNN algorithm.**
2. **Windowing:**
   * **We define a window size of 30 days (window\_size). This window size is used to create input sequences (X) and their corresponding target values (y) by sliding over the dataset. By capturing temporal dependencies within a fixed window, the model can learn patterns in the time series data.**
3. **Train-Test Split:**
   * **We split the dataset into training and testing sets using the train\_test\_split function from sklearn.model\_selection. The training set contains 80% of the data (X\_train, y\_train), while the testing set contains the remaining 20% (X\_test, y\_test). This allows us to evaluate the model's performance on unseen data and detect overfitting.**
4. **Hyperparameter Tuning:**
   * **We specify the number of neighbors (k) as 5 when initializing the KNN model (KNeighborsRegressor). The choice of k can significantly impact the model's performance. By experimenting with different values of k, we can find the optimal balance between bias and variance in the model.**
5. **Inverse Transformation:**
   * **After making predictions using the KNN model (predictions\_knn), we perform an inverse transformation using scaler.inverse\_transform. This step converts the scaled predictions back to the original scale of the closing prices (predictions\_knn). Inverting the scaling transformation allows us to interpret the predictions in the context of the original data and evaluate the model's accuracy.**

**GRADIENT BOOSTING**

**1. Increased Model Complexity: - Initially, a basic Gradient Boosting model with shallow trees and a limited number of estimators was utilized.**

**- To enhance model complexity, the number of boosting stages (estimators) was increased from 100 to 500. This expansion allows the model to iteratively learn from residuals and build more expressive decision trees, capturing intricate patterns and relationships within the data.**

**- Additionally, the maximum depth of each decision tree was adjusted from 3 to 6, enabling the trees to grow deeper and potentially capture more nuanced interactions among features.**

**2. Hyperparameter Tuning:**

**- Parameters such as learning rate (eta), regularization parameters (lambda and alpha), and batch size were fine-tuned using techniques like grid search before we used random search.**

**3.Ensemble Methods:**

**- Gradient Boosting can be combined with other ensemble techniques such as bagging or stacking. By leveraging the diversity of multiple models, ensemble methods can enhance predictive performance and robustness, especially when individual models exhibit complementary strengths and weaknesses.**

**- Advanced gradient boosting variants such as XG-Boost, or Cat-Boost were explored, as they offer optimizations and additional features that can further boost model performance compared to traditional gradient boosting.**

**4.Evaluation and Analysis:**

**- Model performance was evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score. These metrics provide comprehensive insights into the model's predictive capabilities across different aspects, facilitating a thorough assessment of its effectiveness.**

**- Techniques like cross-validation were employed to obtain reliable estimates of model performance and ensure its generalization to unseen data. By rigorously analyzing model outputs and diagnostic plots, valuable insights into model behavior and areas for improvement were obtained, guiding further iterations and refinements.**

**By implementing these strategies and leveraging the flexibility and power of gradient boosting, significant improvements in predictive performance can be achieved, potentially surpassing previous research papers on the dataset. Adjusting the batch size along with other hyperparameters can contribute to better convergence and overall model performance.**