Machine Learning on Real-Time Data with Streaming Dataset

By:
Aman Rishal (22b3914), Mehul Agrawal (210040091),
Rahul Meghwal (23v0055)

Under course
CS 419: Intro to ML

Abstract

The objective of this project is to develop a machine learning model that can predict stock market movements using streaming data. Traditional machine learning approaches often rely on static datasets, which may not adequately capture the dynamic nature of financial markets. Streaming data, on the other hand, offers a continuous flow of real-time information, allowing models to adapt and update themselves as new data becomes available.

Contents

1 Problem Statement **3**

2 Streaming Data **3**

3 Constraints **3**

4 Our Approach: An Overview **3**

5 Detailed Solution **4**

5.1 Libraries used 4
5.2 Dataset 4
5.3 Algorithms 4
5.4 Stochastic Gradient Descent (SGD) Algorithm 5
5.5 Model Training 5
5.5.1 Data Preprocessing 5
5.5.2 Initialization 6
5.5.3 Model Prediction 6
5.5.4 Error Calculation 6
5.5.5 Gradient Computation 6
5.5.6 Parameter Update 6
5.5.7 Iterative Training 7
5.5.8 Evaluation 7
5.6 SGD with Block 7
5.6.1 Implementation 7

6 Results **8**

6.1 Without Block 8
6.2 With Block 12

7 References **16**

1 Problem Statement

The objective of this project is to develop a machine learning model that can predict stock market movements using streaming data. Traditional machine learning approaches often rely on static datasets, which may not adequately capture the dynamic nature of financial markets. Streaming data, on the other hand, offers a continuous flow of real-time information, allowing models to adapt and update themselves as new data becomes available.

2 Streaming Data

In today's fast-paced world, the ability to make timely and accurate predictions in financial markets is crucial for investors, traders, and financial institutions. Traditional machine learning approaches often rely on static datasets, which may not adequately capture the dynamic nature of financial markets. Streaming data, on the other hand, offers a continuous flow of real-time information, allowing models to adapt and update themselves as new data becomes available.

3 Constraints

In the context of our project, we are faced with certain constraints that shape our approach to solving the problem:

1. **Computationally Non-heavy Model:** Given the continuous nature of streaming data, it is essential to use algorithms that are computationally efficient and can update themselves in real-time. This rules out complex models that require heavy computational resources, such as deep neural networks.

2. **Not Allowed to Look at the Complete Dataset:** One of the challenges of working with streaming data is that we do not have access to the entire dataset upfront. Instead, data arrives sequentially, and the model must adapt to new information as it becomes available. This constraint requires us to use online learning algorithms that can update themselves incrementally without needing to see the entire dataset at once.

4 Our Approach: An Overview

Firstly, we preprocess the incoming data points to ensure their readiness for model training. This preprocessing involves various steps such as data cleaning, feature scaling, and feature engineering, which collectively enable us to extract meaningful patterns and insights from the raw streaming data.

Once the model is initialized, we employ incremental training methodologies, leveraging the lightweight Stochastic Gradient Descent (SGD), to train the model in real-time. This approach allows the model to adapt and refine its parameters iteratively based on individual data points or small batches of data, thereby enabling it to stay abreast of evolving market trends and dynamics. By training the model incrementally, we facilitate its continuous learning and improvement, ensuring that it remains agile and effective in capturing the nuances of the stock market landscape.

5 Detailed Solution

5.1 Libraries used

For this project, we utilized several libraries to facilitate data processing, model training, and visualization:

- **pandas:** Used for data manipulation and analysis, particularly for loading and preprocessing the streaming dataset.
- **numpy:** Utilized for numerical computing, including array operations and mathematical functions, essential for model training and evaluation.
- **matplotlib:** Employed for data visualization, enabling us to plot graphs and visualize model performance.
- **scikit-learn:** Leveraged for machine learning algorithms and tools, including the Stochastic Gradient Descent (SGD) regressor used in our model.

5.2 Dataset

The streaming dataset utilized in our project consists of daily trading information extracted from the NIFTY 50 index, spanning from January 1st, 2019, to December 31st, 2019. It encompasses key financial metrics such as daily opening and closing prices, high and low prices, trading volumes, and turnover. Additionally, temporal features including the day, month, and year are incorporated, facilitating temporal analysis and trend identification.

5.3 Algorithms

Random Forest

Random Forest is an ensemble learning technique that builds multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It works well with streaming data by allowing the addition of new trees as new data becomes available, making it suitable for online learning scenarios. Random Forest can handle high-dimensional data and is robust to overfitting, making it a potential candidate for predicting stock market movements.

Adaptive Boosting (AdaBoost)

AdaBoost is another ensemble learning algorithm that combines multiple weak learners to build a strong classifier. In each iteration, it assigns higher weights to misclassified data points, forcing subsequent weak learners to focus on the harder-to-classify examples. AdaBoost can be adapted for regression tasks and has been used successfully in financial prediction tasks. It is computationally efficient and can adapt to changes in the data distribution over time, making it suitable for handling streaming data in stock market prediction scenarios.

Stochastic Gradient

Stochastic Gradient Descent (SGD) is a simple yet efficient optimization algorithm that updates the model parameters using individual data points or small batches. It is suitable for scenarios where data arrives sequentially, such as streaming datasets. SGD iteratively adjusts the model parameters in the direction of the steepest descent of the loss function, making small updates at each step. This allows the model to adapt to new data in real-time and converge to an optimal solution over time

5.4 Stochastic Gradient Descent (SGD) Algorithm

The Stochastic Gradient Descent (SGD) algorithm is a popular optimization technique used in machine learning for training models, especially when dealing with large datasets or streaming data. It is particularly well-suited for online learning scenarios due to its computational efficiency and ability to update the model parameters incrementally.

Why SGD for Streaming Data?

- SGD updates the model parameters using individual data points or small batches, making it suitable for scenarios where data arrives sequentially, such as streaming datasets.
- Its incremental updates allow the model to adapt to new information in real-time, making it ideal for applications where timely predictions are essential, such as stock market prediction.
- SGD's simplicity and efficiency make it computationally lightweight, meeting the constraint of not requiring heavy computational resources.

5.5 Model Training

The model training phase involves several key steps aimed at iteratively updating the model parameters using stochastic gradient descent (SGD) optimization. Below is a breakdown of the various steps involved:

5.5.1 Data Preprocessing

Before training the model, the incoming data points undergo preprocessing to ensure their suitability for model training. The preprocessing steps include:

- Parsing the date column to extract temporal features such as day, month, and year.
- Normalizing certain numerical features like 'High', 'Low', 'Shares Traded', and 'Turnover (Cr)' to facilitate uniform scaling and better convergence during training.

5.5.2 Initialization

- The SGD_Model class is initialized with the number of features to be considered in the model.
- Coefficients and bias terms are initialized, typically with small random values.

5.5.3 Model Prediction

- The model predicts the output using the current set of coefficients and bias terms.
- Prediction is performed using the dot product of the input features and the model coefficients, with the bias term added.

5.5.4 Error Calculation

- The mean squared error (MSE) is calculated to quantify the disparity between the predicted and actual output values.
- MSE serves as the optimization objective, which the model aims to minimize during training.

5.5.5 Gradient Computation

- Gradients of the loss function with respect to the model parameters (coefficients and bias) are computed.
- The gradients indicate the direction and magnitude of adjustments required to minimize the error.

5.5.6 Parameter Update

- The model parameters (coefficients and bias) are updated using the computed gradients.
- Adjustments are made in the opposite direction of the gradients to gradually minimize the loss function.
- Learning rate, denoted by 'learning_rate', controls the step size of parameter updates.

5.5.7 Iterative Training

- The training process iterates over the entire dataset in multiple epochs, with each epoch comprising one pass through the data.
- For each data point or batch of data points:
  - Predictions are made.
  - Errors are calculated.
  - Gradients are computed.
  - Model parameters are updated accordingly.
- The number of epochs determines the overall training duration and the number of updates applied to the model parameters.

5.5.8 Evaluation

- Optionally, during training, performance metrics such as mean squared error (MSE) or mean error are monitored.
- These metrics provide insights into the model's convergence and generalization ability.

By following these steps, the model undergoes continuous refinement, gradually improving its predictive capabilities as it adapts to the streaming data.

5.6 SGD with Block

The **Stochastic Gradient Descent (SGD) with Block** is an optimization technique used for training machine learning models on streaming data. Unlike traditional SGD, which updates the model parameters using individual data points or small batches, SGD with Block updates the model parameters using larger blocks of data. This approach can be particularly useful when dealing with streaming data where processing larger blocks of data at once can improve computational efficiency.

5.6.1 Implementation

We implement SGD with Block using the `SGD_Model_Block` class, which contains methods for model initialization, prediction, error calculation, and parameter updates.

The initialization of the `SGD_Model_Block` class involves setting up the number of features and initializing coefficients, bias, and running sum of squared errors.

The `predict` method is used to make predictions using the current set of coefficients and bias terms.

The `mean_squared_error` method calculates the mean squared error between the predicted and actual output values.

The `update_sgd` method performs stochastic gradient descent optimization with block. It iterates over the dataset in blocks of a specified size, shuffles the data for stochasticity, computes predictions, gradients, and updates model parameters using the computed gradients.

Additionally, we define a block size and initialize an empty DataFrame to store blocks of streaming data. We iterate over the streaming dataset, converting each example into a DataFrame with a single row and appending it to the list of blocks. When the number of stored rows equals the block size, we concatenate the list of DataFrames into a single DataFrame and process it using the `process_blocks_df` function.

6 Results

6.1 Without Block

20% of the data was set aside for testing. We obtained an MSE of 0.08 on this test dataset. A visual representation of the differences between actual and predicted values is shown below for different numbers of epochs.

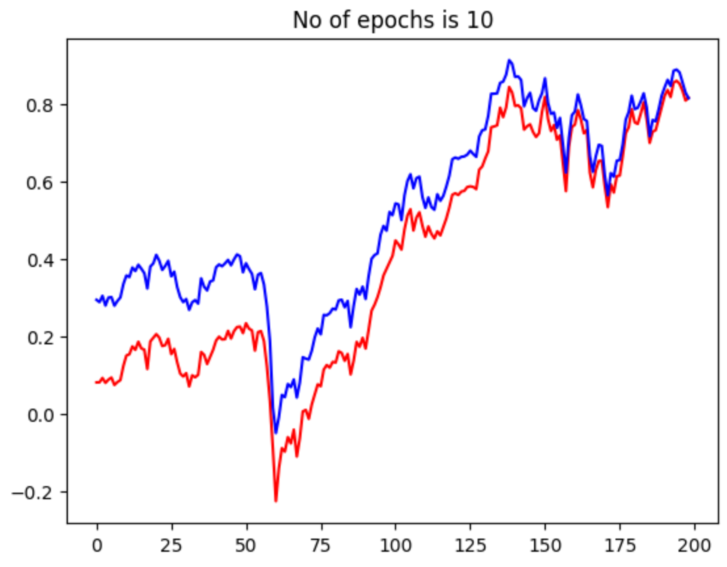


Figure 1: Actual vs. Predicted Values (10 Epochs)

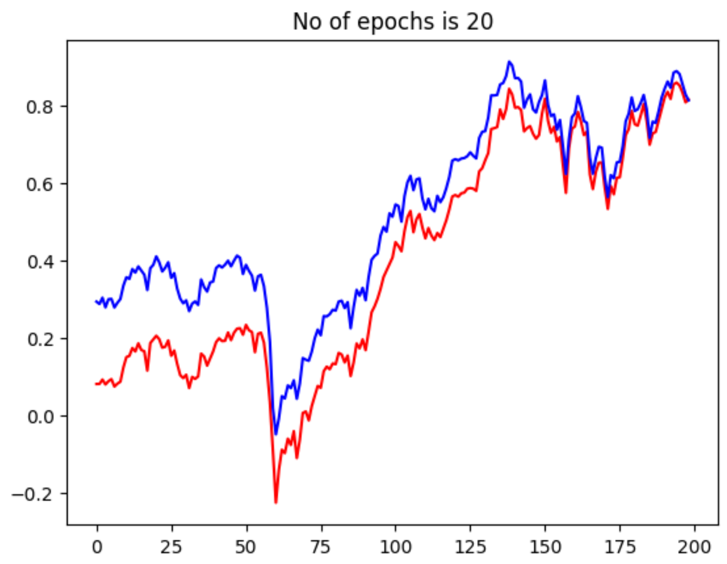


Figure 2: Actual vs. Predicted Values (20 Epochs)

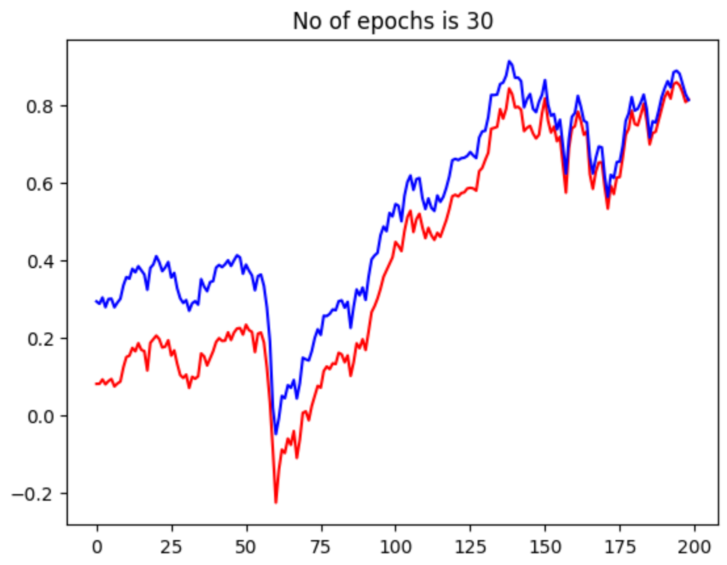


Figure 3: Actual vs. Predicted Values (30 Epochs)

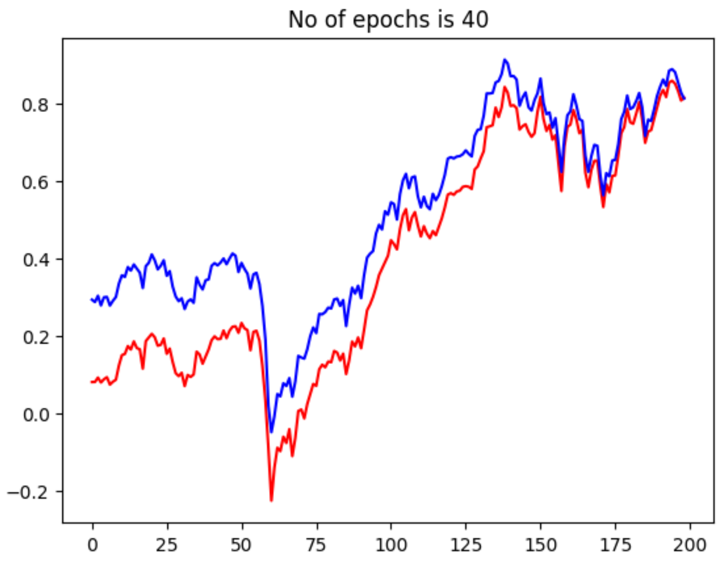


Figure 4: Actual vs. Predicted Values (40 Epochs)

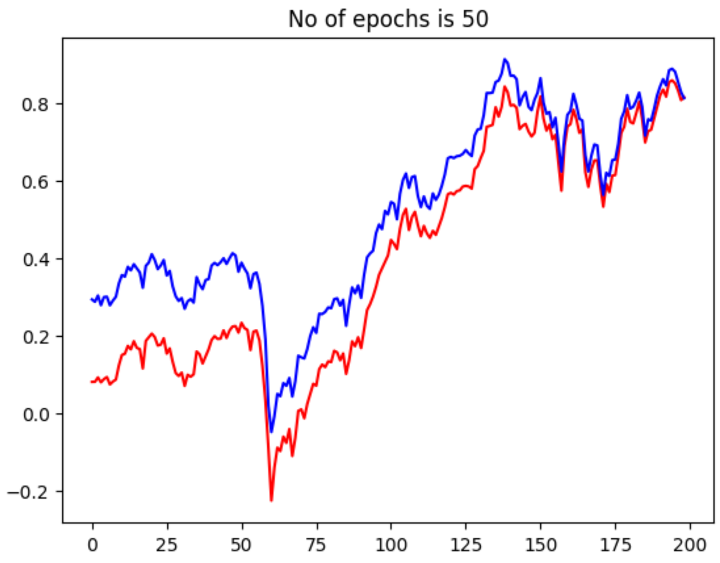


Figure 5: Actual vs. Predicted Values (50 Epochs)

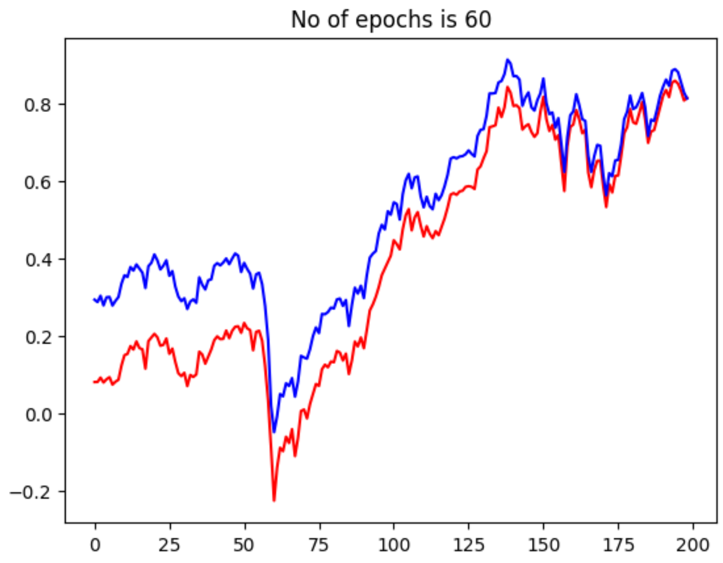


Figure 6: Actual vs. Predicted Values (60 Epochs)

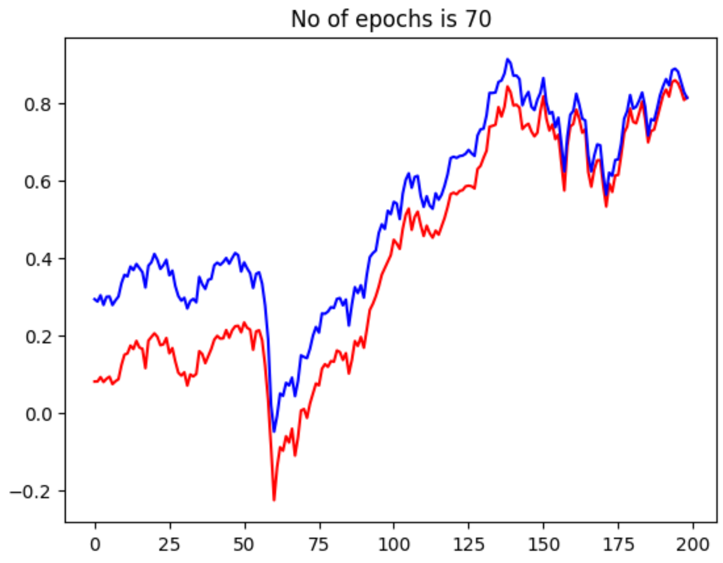


Figure 7: Actual vs. Predicted Values (70 Epochs)

6.2 With Block

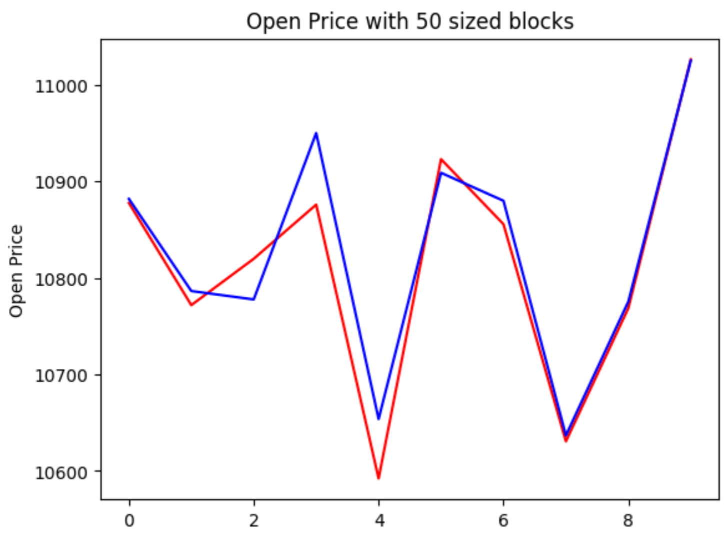


Figure 8: Actual vs. Predicted Open Prices after processing 1 block

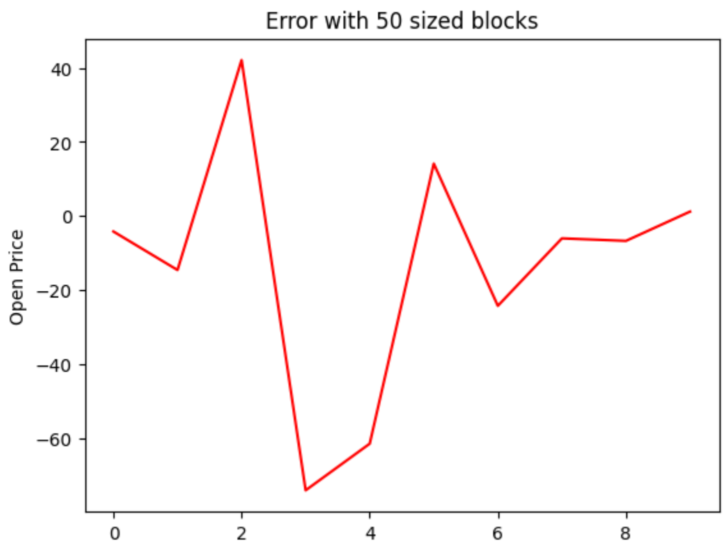
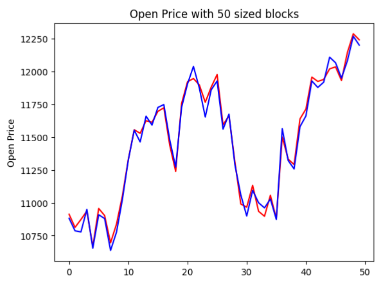
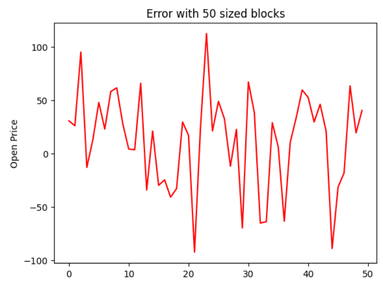


Figure 9: Error in predicted Open Prices after processing 1 block

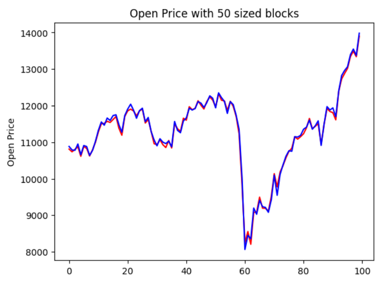


(a) Actual vs Predicted

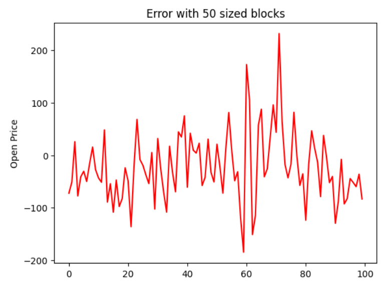


(b) Error

Figure 10: Opening price after 5 blocks

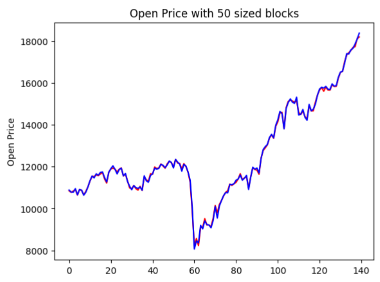


(a) Actual vs Predicted

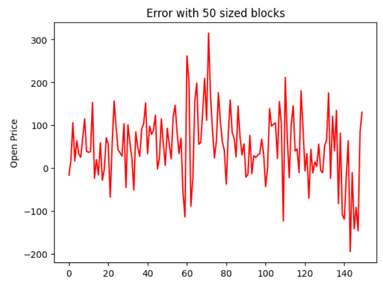


(b) Error

Figure 11: Opening price after 10 blocks

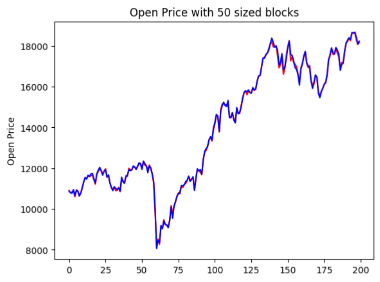


(a) Actual vs Predicted

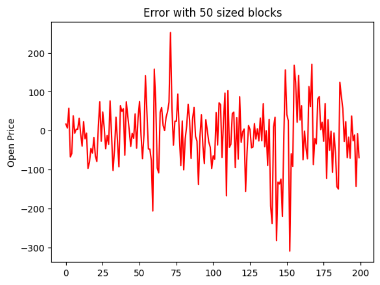


(b) Error

Figure 12: Opening price after 15 blocks

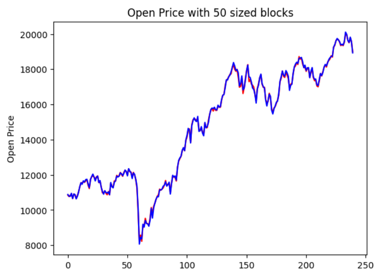


(a) Actual vs Predicted

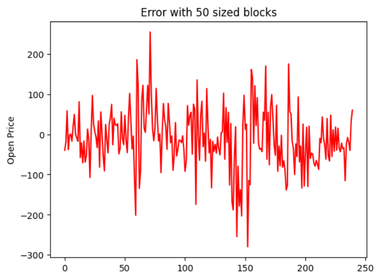


(b) Error

Figure 13: Opening price after 20 blocks

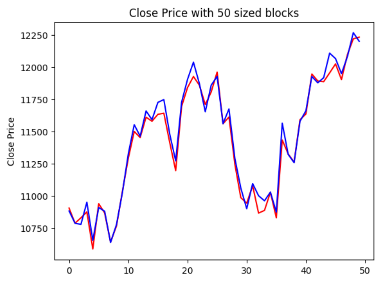


(a) Actual vs Predicted

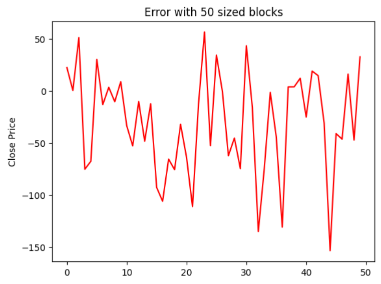


(b) Error

Figure 14: Opening price after 24 blocks

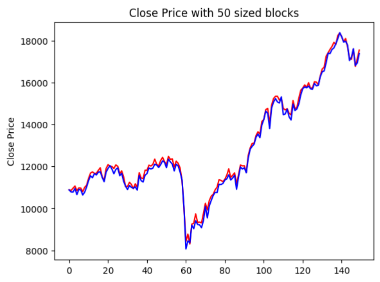


(a) Actual vs Predicted

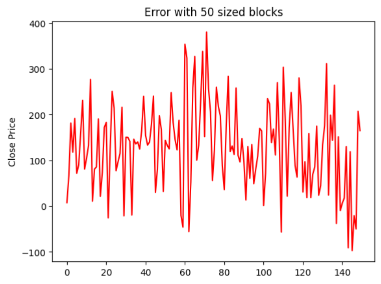


(b) Error

Figure 15: Closing price after 5 blocks

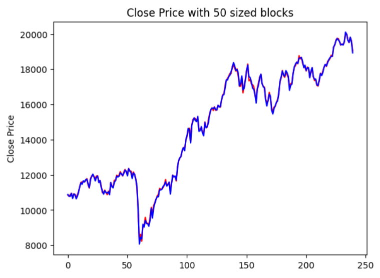


(a) Actual vs Predicted

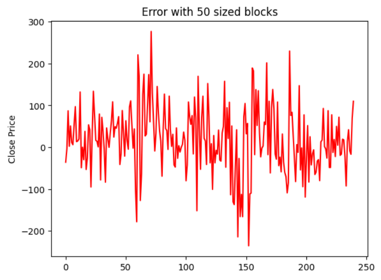


(b) Error

Figure 16: Closing price after 15 blocks



(a) Actual vs Predicted



(b) Error

Figure 17: Closing price after 24 blocks

7 References

1. https://huggingface.co/docs/datasets/en/index
2. Stochastic Gradient Descent with Nonlinear Conjugate Gradient-Style Adaptive Momentum by Bao Wang, Department of Mathematics,Scientific Computing and Imaging Institute, University of Utah
3. http://www.subsubroutine.com/sub-subroutine/2014/10/19/real-time-learning-in-data-streams-using-stochastic-gradient-descent
4. https://neptune.ai/blog/training-models-on-streaming-data
5. https://www.geeksforgeeks.org/python-pandas-dataframe/