Project Report

Backtesting and Risk Assessment for Financial Institutions

CS354-N - Computational Intelligence

Course Instructor: Dr. Aruna Tiwari

Abstract

This project uses a Long Short-Term Memory (LSTM) network to forecast Bitcoin prices based on historical market data. The model is trained on normalized time-series inputs and evaluated using MAE, RMSE, and R^2 score, showing strong predictive accuracy.

We simulate a trading strategy using a Moving Average Crossover approach on the predicted prices. Backtesting with different window sizes and financial metrics such as Total Return, Volatility, Sharpe Ratio, and Max Drawdown reveals that while the model predicts well, trading outcomes depend heavily on the strategy used.

This work highlights the potential of combining deep learning with financial backtesting for informed decision-making in crypto markets.

Introduction

Bitcoin is a highly volatile and unpredictable asset, making its price forecasting a complex task. Traditional models often fail to capture the dynamic nature of the crypto market and are rarely evaluated in real trading scenarios.

To address this, our project uses a Long Short-Term Memory (LSTM) neural network to predict Bitcoin prices based on historical market data. The model is trained to learn temporal patterns and is evaluated using real-world backtesting.

We simulate a trading strategy based on these predictions and assess performance using financial metrics like Total Return, Sharpe Ratio, and Max Drawdown. This combination of prediction and backtesting allows us to evaluate both accuracy and practical utility.

The goal is to create a reliable, machine learning-based framework that can support financial decision-making in volatile markets.

3. Dataset

The dataset used in this project consists of historical Bitcoin price data obtained via the Deribit API. This dataset includes key trading features such as Open, High, Low, Close, and Volume for each trading interval. For effective modeling, we used a 50-minute interval to

create consistent time steps that capture short-term market behavior without introducing excessive noise.

Before feeding the data into the model, several preprocessing steps were performed:

- **Missing Value Handling**: Any missing or corrupted data points were cleaned to ensure smooth training.
- **Normalization**: We applied `MinMaxScaler` from `sklearn.preprocessing` to scale all features between 0 and 1. This ensures that the LSTM model is not biased by large numeric differences in feature values.
- **Sequence Construction**: Input sequences of 50 time steps were constructed to allow the model to learn from recent historical trends.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(raw_data)
```

4. Model Architecture

The model used in this project is based on a Sequential architecture, leveraging the strengths of Long Short-Term Memory (LSTM) networks for time series forecasting. LSTM networks are particularly well-suited for sequential data because they are designed to capture long-term dependencies and mitigate the vanishing gradient problem found in traditional RNNs.

Layer-by-Layer Explanation

1. **First LSTM Layer (64 units, return_sequences=True)**
    a. This layer processes the input sequences and extracts temporal features.
    b. `return_sequences=True` ensures the full output sequence is passed to the next LSTM layer.
2. **Dropout Layer (rate=0.2)**
    a. Applied to reduce overfitting by randomly setting 20% of the units to zero during training.
3. **Second LSTM Layer (64 units, return_sequences=False)**
    a. Takes the feature representations from the previous LSTM layer and compresses them to a fixed-length vector.
    b. `return_sequences=False` ensures only the final output is passed forward.
4. **Dropout Layer (rate=0.2)**
    a. Additional regularization to further mitigate overfitting risks.

5. **Dense Layer (32 units, ReLU activation)**
    a. Fully connected layer that helps capture non-linear combinations of features.
    b. ReLU activation introduces non-linearity into the model.
6. **Output Dense Layer (1 unit)**
    a. Final layer that outputs the predicted closing price of Bitcoin.

Compilation Details

- **Loss Function**: Mean Squared Error (MSE), appropriate for regression tasks.
- **Optimizer**: Adam, chosen for its adaptive learning rate capabilities and good convergence behavior.

```
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(time_steps,
X_seq.shape[2])),
    Dropout(0.2),
    LSTM(64, return_sequences=False),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

5. Model Training and Evaluation

The LSTM model was trained using the normalized and sequenced historical data, where each training sample consisted of 50 previous time steps to predict the next closing price of Bitcoin.

Data Splitting

- The dataset was divided into **80% for training** and **20% for testing**.
- Training was performed over multiple epochs with mini-batch processing to ensure convergence without overfitting.

Training Process

- The model was compiled using the **Adam optimizer**, known for adaptive learning rate capabilities.
- The loss function was **Mean Squared Error (MSE)**, suitable for continuous-valued regression targets.
- To prevent overfitting, **Dropout layers** were added after each LSTM block.

```
history = model.fit(X_seq, Y_seq, epochs=50, batch_size=32,

validation_split=0.1, verbose=1)
```

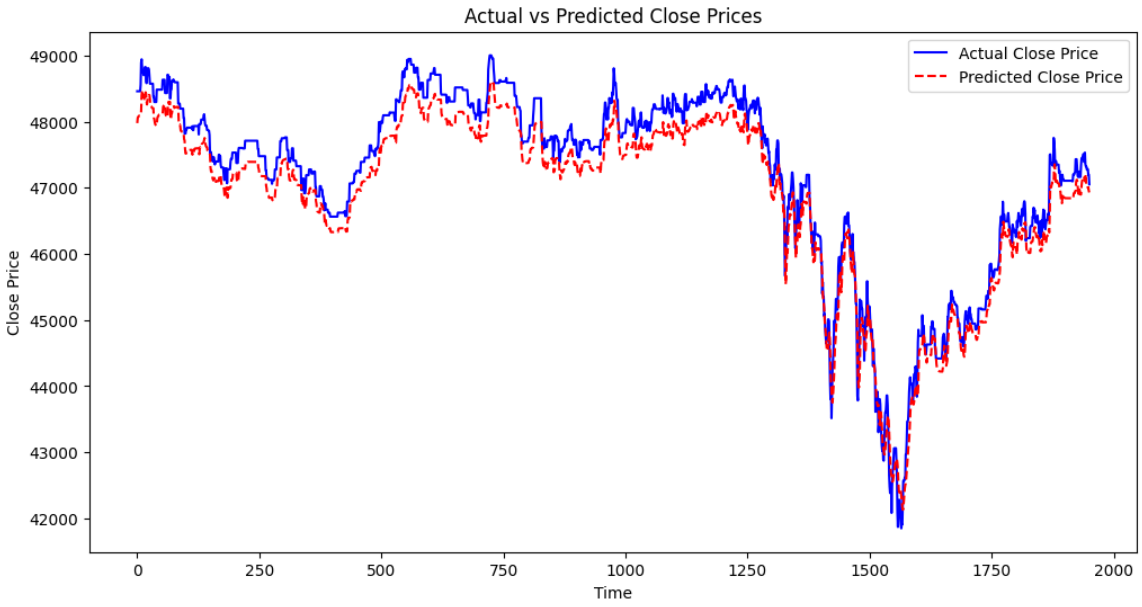
The training process was monitored using the validation loss to ensure the model was learning generalizable patterns and not memorizing the training data.

Prediction and Evaluation

- After training, the model was used to predict prices on the test set.
- The predicted values were inverse-transformed to obtain original price scale predictions.
- Evaluation was performed using the following metrics:
    - **Mean Absolute Error (MAE)** – Measures average prediction error.
    - **Root Mean Squared Error (RMSE)** – Penalizes larger errors more heavily.
    - **R^2 Score** – Indicates how well the predictions explain the variance in actual prices.
    - **Mean Absolute Percentage Error (MAPE)** – Shows prediction error as a percentage of actual value.

Results

Predicted vs Actual Prices Graph



Evaluation Metrics Table

Metric	Value
MAE	307.30
RMSE	329.42
R^2 Score	0.9469
MAPE	0.65%

6. Backtesting Strategy

Backtesting is a critical step in evaluating the real-world applicability of a trading strategy. It involves simulating trades based on historical data using the predictions of the model to assess how well the strategy would have performed. In this project, we implemented a **Moving Average Crossover** strategy using the predicted Bitcoin prices generated by our LSTM model.

Strategy Overview: Moving Average Crossover

The **Moving Average Crossover** strategy is a simple yet widely used technical trading approach. It is based on the principle that the price trend can be inferred by comparing a short-term moving average to the price or another longer-term average.

For our implementation, we define the strategy as follows:

- **Buy Signal**: If the predicted price crosses *above* its moving average, it is interpreted as a bullish signal, and the strategy initiates a **buy** action.
- **Sell Signal**: If the predicted price crosses *below* its moving average, it is interpreted as a bearish signal, prompting a **sell** action.

This technique attempts to exploit trend momentum, if an upward crossover indicates a forthcoming price increase, and vice versa.

Implementation Details

The backtest simulation operates under the following assumptions:

- Starts with a fixed initial capital (e.g., $10,000).
- Tracks cash and Bitcoin holdings.
- Executes trades based on crossover signals.
- Computes the total portfolio value at each time step.

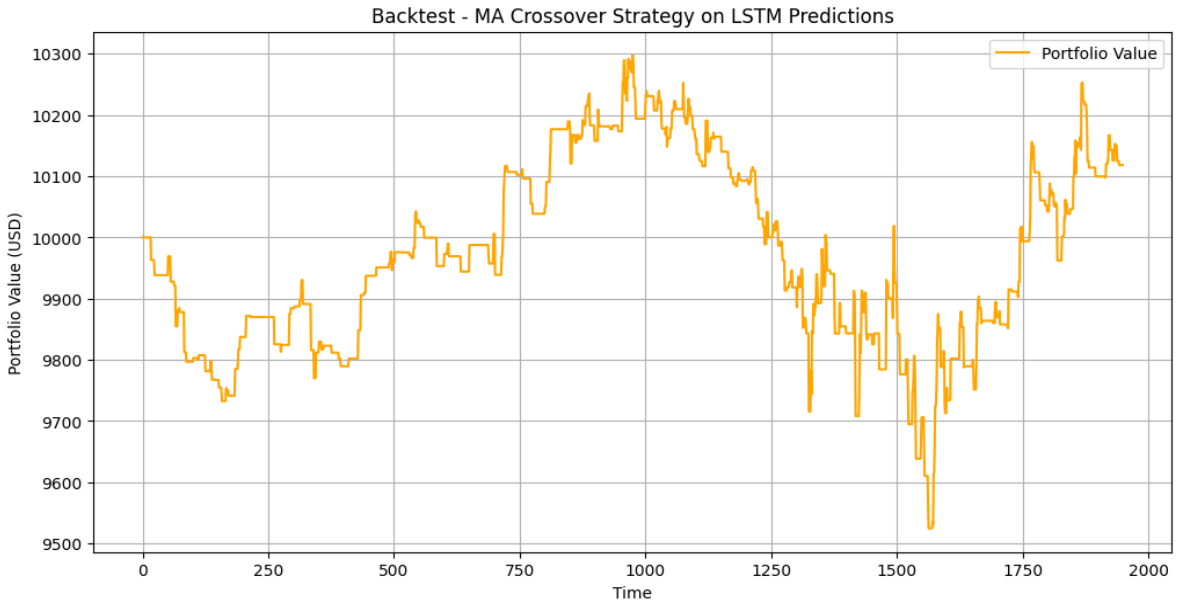
We calculate the **rolling moving average** of the predicted prices over a defined window. The trading logic is evaluated at each time step based on the relationship between the predicted price and its moving average.

```
def backtest_with_moving_avg(actual_prices, predicted_prices, window,
initial_cash):
```

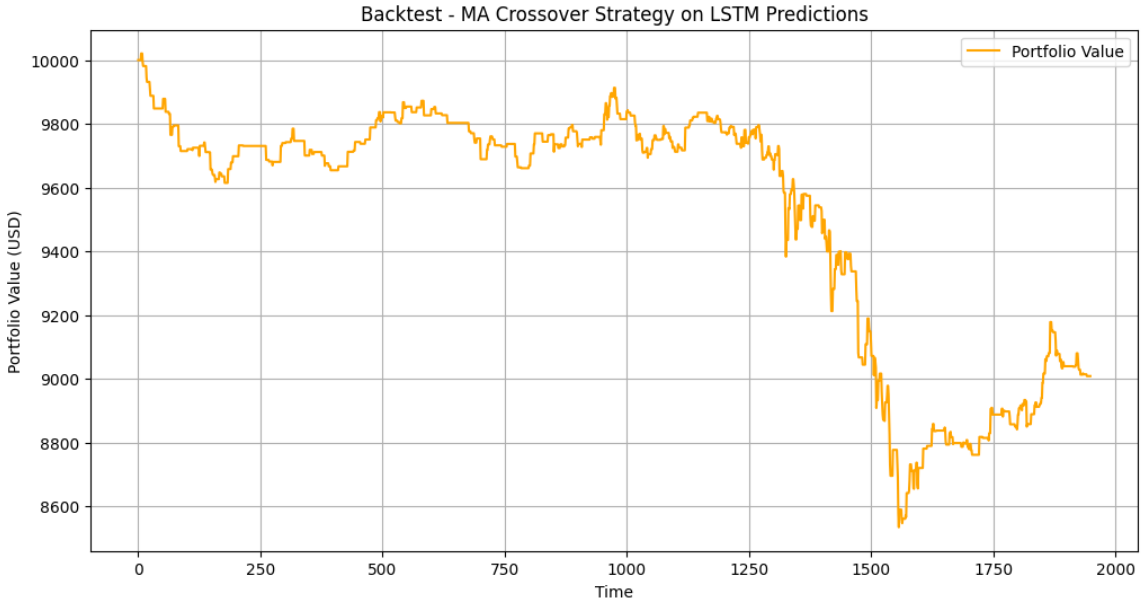
Observations

The following plots illustrate the simulated portfolio value over time when applying the Moving Average Crossover strategy on LSTM predictions.

Window Size = 2



Window Size = 2



7. Risk Assessment

Accurate price prediction is only one part of building an effective trading system. Equally important is understanding the **risk profile** of the resulting strategy. In this project, we assessed risk using several industry-standard metrics, applied to the simulated portfolio generated from backtesting the LSTM-driven Moving Average Crossover strategy.

Key Risk Metrics

To evaluate the risk-adjusted performance of the strategy, we calculated the following:

1. **Volatility**
   Volatility measures the standard deviation of portfolio returns. It indicates how much the portfolio value fluctuates over time. High volatility generally suggests higher uncertainty and risk.
2. **Sharpe Ratio**
   The Sharpe Ratio quantifies return per unit of risk. A positive Sharpe Ratio implies returns are favorable relative to volatility, while a negative value indicates that the strategy is underperforming even after accounting for risk.
3. **Maximum Drawdown (MDD)**
   This is the largest observed loss from a peak to a trough in portfolio value. A high drawdown implies the strategy may expose investors to significant losses before recovering, which can affect investor confidence and capital preservation.

Metric Values (Window Sizes 2 and 3)

The table below summarizes the performance of the Moving Average Crossover strategy using different window sizes.

Window Size	Final Portfolio Value	Total Return	Volatility	Sharpe Ratio	Max Drawdown
2	$10,118.27	+1.18%	0.0016	0.0046	$772.82
3	$9,007.96	-9.92%	0.0017	-0.0305	$1,487.03

Limitations & Risk Factors

- **No Transaction Costs**: The strategy currently ignores trading fees, slippage, and market liquidity constraints, which would further reduce real-world returns and increase drawdowns.
- **Overfitting Risk**: Since the model was trained and backtested on the same data source, there's a risk of overfitting. Real market dynamics may differ significantly.

8. Conclusion

In this project, we successfully developed a machine learning-based forecasting and risk assessment pipeline tailored for financial market analysis, specifically targeting Bitcoin price prediction and strategy evaluation.

- We designed and implemented an **LSTM-based neural network** capable of learning from historical market data to accurately forecast short-term Bitcoin closing prices. The model achieved strong performance across several evaluation metrics, including a high R^2 score and low mean absolute error, validating its suitability for time series prediction tasks.
- To evaluate the practical usability of these predictions, we **integrated a backtesting framework** that simulates real-world trading using a **Moving Average Crossover** strategy. This framework enabled us to analyze the outcomes of trading decisions based solely on the model's predictions.
- Our analysis revealed a key insight: while the **forecasting model was reliable**, the **trading logic was relatively simplistic**. The profitability and risk profile varied significantly with the choice of strategy parameters (e.g., moving average window size), emphasizing the need for more refined and adaptive trading rules.
- To complement the performance analysis, we incorporated **risk assessment metrics** such as **Volatility, Sharpe Ratio**, and **Maximum Drawdown**. These provided a deeper understanding of how much risk was taken to achieve the observed returns, and highlighted the limitations of the current strategy in terms of capital preservation and consistency.

In summary, the project demonstrated a strong foundation in predictive modeling and quantitative evaluation. However, it also underscored the critical importance of pairing accurate predictions with robust, risk-aware trading strategies.

9. Future Work

To build upon the current work, several enhancements can be pursued:

- **External Data Integration**: Enhance prediction accuracy by including external factors such as news sentiment, economic indicators, or on-chain data.
- **Real-Time System**: Develop a real-time prediction and trading system using live market feeds and automated execution through APIs.
- **Model Improvements**: Explore advanced architectures like Transformers or ensemble models to improve forecasting performance.

These enhancements aim to bridge the gap between academic modeling and practical, risk-aware trading applications.