Multi-Coin Logistic Regression Project

Using Binance Order Book Data

Aamer Jalan
Arin Baswana

February 25, 2025

Abstract

This document details an end-to-end machine learning project that aims to predict *short-term* (10–20 tick horizon) price movements (*up*, *neutral*, *down*) for multiple cryptocurrencies traded on the Binance exchange. All data collection, cleaning, processing, and modeling steps are implemented using only `pandas` and `NumPy`. Our model of choice is multi-class logistic regression (softmax regression), due to its interpretability, straightforward multi-class formulation, and alignment with the supervised learning methods taught in our course.

Contents

1 Introduction

Algorithmic and high-frequency traders increasingly rely on sophisticated data-driven techniques to parse real-time order book information. In this project, we construct a *supervised learning pipeline* that uses order book data from the Binance API to predict the short-term price direction (up, neutral, or down) of multiple crypto assets—including BTC, ETH, BNB, XRP, and LTC—over a 10–20 tick horizon.

A key aspect of this project is that it is built solely with `pandas` and `NumPy`; no external machine learning libraries (e.g., `scikit-learn`, `TensorFlow`, `PyTorch`) are employed. This ensures we fully understand and implement each data processing and modeling step: from feature extraction and labeling, to the mathematical derivation and gradient-based training of a multi-class logistic regression model.

2 Project Goal

The **primary goal** is to develop a model that classifies short-term price movements of selected cryptocurrency pairs (BTCUSDT, ETHUSDT, BNBUSDT, XRPUSDT, LTCUSDT) into one of three classes:

$$\text{Class} = \begin{cases} +1 & \text{price up,} \\ 0 & \text{price neutral,} \\ -1 & \text{price down.} \end{cases}$$

The predictions are made at a horizon of 10–20 ticks (or time steps) in the future.

We aim to:

1. **Collect:** Continuously fetch raw order book snapshots from the Binance API.
2. **Clean and Process:** Parse and consolidate the data into a time-aligned feature matrix for each cryptocurrency.

3. **Engineer Features:** Extract relevant signals (mid-price, spread, volume imbalance, rolling volatility, etc.) and incorporate cross-asset features for multi-coin interactions (e.g., how BTC's movements affect altcoins).

4. **Train Model:** Implement multi-class logistic regression from scratch in `NumPy`.

5. **Evaluate:** Split the data chronologically into training/testing sets and measure accuracy, precision, recall, and other metrics of interest.

3 Data Collection: Binance API

3.1 Asset Selection

We choose five cryptocurrency pairs:

$$\{\text{BTCUSDT}, \text{ETHUSDT}, \text{BNBUSDT}, \text{XRPUSDT}, \text{LTCUSDT}\}.$$

These pairs are heavily traded and exhibit substantial liquidity, making them good candidates for high-frequency data analysis.

3.2 API Endpoint and Method

- **API Endpoint:**

  `https://api.binance.com/api/v3/depth?symbol={symbol}&limit=1000`.

- **Asynchronous Fetching:** We use Python's `asyncio` and `aiohttp` to concurrently fetch order book snapshots for all chosen assets. This helps minimize latency and ensures near-simultaneous snapshots.

- **Storage:** The raw JSON responses are saved (e.g., `data/raw/raw_crypto_orderbooks.json`), tagged with timestamps and asset symbols. Each record includes bids, asks, and volumes at multiple price levels.

4 Data Cleaning and Processing

4.1 Parsing Order Book Data

We convert raw JSON objects into `pandas DataFrames`, ensuring each exchange snapshot can be *uniquely identified* by:

- `timestamp` (in milliseconds).

- `asset_symbol` (e.g., `BTCUSDT`).

- `bid_prices`, `ask_prices`, `bid_volumes`, `ask_volumes`.

4.2 Feature Engineering

The following features are computed for each snapshot:

1. **Mid-Price:**
$$\text{MidPrice} = \frac{\text{BestBid} + \text{BestAsk}}{2}.$$

2. **Spread:**
$$\text{Spread} = \text{BestAsk} - \text{BestBid}.$$

3. **Volume Imbalance:**
$$\text{Imbalance} = \frac{\text{BidVolume} - \text{AskVolume}}{\text{BidVolume} + \text{AskVolume} + \epsilon},$$
where ϵ is a small constant to avoid division by zero.

4. **Rolling Volatility (Optional):**
$$\sigma_{roll} = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(p_j - \overline{p})^2},$$
computed on the mid-price over a rolling window (e.g., 1 second).

4.3 Cross-Asset Alignment

Since we want to incorporate **cross-asset** features, we align timestamps across all five cryptocurrencies. For each timestamp t, we build a *concatenated feature vector* $\mathbf{x}_t$ that includes:

$$\underbrace{\left[\text{MidPrice}_{\text{asset}}, \text{Spread}_{\text{asset}}, \text{Imbalance}_{\text{asset}}, \sigma_{\text{asset}}\right]}_{\text{Own-Asset Features}} \quad \text{plus} \quad \underbrace{\left[\text{MidPrice}_{\text{BTC}}, \text{Spread}_{\text{BTC}}, \ldots\right]}_{\text{Cross-Asset: BTC}}$$

This is repeated for ETH, BNB, XRP, LTC if relevant.

4.4 Processed Data Storage

Finally, each asset's per-timestamp feature vector is stored in a CSV (e.g., `data/processed/processed_dat` for downstream modeling.

5 Train-Test Split

We use a **time-based** split: the first 70% of chronological data is for training, and the final 30% is for testing. This approach prevents look-ahead bias and mirrors real-world trading scenarios where future data is never known at training time.

5.1 Label Generation

For each asset and time t, we compute the **future percentage change** of the mid-price over a 10–20 tick horizon:

$$\Delta p_t = \frac{\text{MidPrice}_{t+h} - \text{MidPrice}_t}{\text{MidPrice}_t},$$

where h is the horizon (in ticks). We discretize Δp_t into three classes:

$$y_t = \begin{cases} +1 & \Delta p_t > \alpha, \\ 0 & -\alpha \leq \Delta p_t \leq \alpha, \\ -1 & \Delta p_t < -\alpha, \end{cases}$$

for some threshold $\alpha > 0$ (e.g., $\alpha = 0.0005$).

6 Logistic Regression Model

We adopt a **multi-class** formulation of logistic regression (often called *softmax regression*).

6.1 Model Definition

Let $K = 3$ be the number of classes (*down, neutral, up*). For each training example $\big(\mathbf{x}^{(i)}, y^{(i)}\big)$, we define:

$$P(y^{(i)} = k \mid \mathbf{x}^{(i)}; \boldsymbol{\Theta}) = \frac{\exp\big(\boldsymbol{\theta}_k^\top \mathbf{x}^{(i)}\big)}{\sum_{j=1}^{K} \exp\big(\boldsymbol{\theta}_j^\top \mathbf{x}^{(i)}\big)},$$

where $\boldsymbol{\theta}_k$ is the parameter vector associated with class k, and $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3]$ collects them all.

6.2 Loss Function

We use the **categorical cross-entropy**:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \mathbf{1}\{y^{(i)} = k\} \ln\Big(P(y^{(i)} = k \mid \mathbf{x}^{(i)}; \boldsymbol{\Theta})\Big).$$

Here, m is the total number of training examples.

6.3 Training via Gradient Descent

We implement the gradient-based update rules in pure `NumPy`:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k - \alpha \, \frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\theta}_k},$$

where α is the learning rate. The gradient is computed by combining the chain rule with partial derivatives of the softmax function.

7 Implementation Details in `NumPy`

7.1 Model Initialization

- **Dimensions**: If each example is in $\mathbb{R}^d$ features, then $\boldsymbol{\Theta}$ is a $d \times K$ matrix (plus a bias term if used).
- **Parameter Initialization**: Usually small random values, e.g., $\boldsymbol{\theta}_k \sim \mathcal{N}(0, \sigma^2)$.

7.2 Forward Pass (Softmax)

$$z_k^{(i)} = \boldsymbol{\theta}_k^\top \mathbf{x}^{(i)}, \quad p_k^{(i)} = \frac{\exp(z_k^{(i)})}{\sum_{j=1}^K \exp(z_j^{(i)})}.$$

7.3 Gradient Computation

If $y^{(i)} \in \{1, 2, 3\}$ is the **class index** for example i, then

$$\frac{\partial J}{\partial \boldsymbol{\theta}_k} = \frac{1}{m} \sum_{i=1}^m \left(p_k^{(i)} - \mathbf{1}\{y^{(i)} = k\}\right) \mathbf{x}^{(i)}.$$

7.4 Implementation Outline

1. **Load Data:** from `processed_data.csv`.
2. **Split into Train/Test:** as in Section 5.
3. **Initialize $\boldsymbol{\Theta}$.**
4. **Loop (Epochs):**
   - Compute the softmax probabilities $p_k^{(i)}$ for all i, k.
   - Calculate the gradient $\nabla J(\boldsymbol{\Theta})$.
   - Update $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \alpha \nabla J$.

8 Evaluation Metrics

We use the final 30% hold-out dataset to quantify generalization. For each asset, we evaluate:

8.1 Accuracy

$$\text{Accuracy} = \frac{\text{\# correct predictions}}{\text{\# total predictions}}.$$

8.2 Confusion Matrix

A 3×3 matrix showing counts of $\{\text{down}, \text{neutral}, \text{up}\}$ predictions vs. true labels. This reveals class-specific performance and possible biases (e.g., always predicting *neutral*).

8.3 Precision, Recall, and F1-Score

We can aggregate metrics using *macro-averaging*:

$$\text{Precision}_{macro} = \frac{1}{3}\sum_{k=1}^{3} \text{Precision}_k, \quad \text{Recall}_{macro} = \frac{1}{3}\sum_{k=1}^{3} \text{Recall}_k.$$

The F1-score is the harmonic mean of precision and recall.

8.4 Additional Reliability Checks

- **Time-based Rolling Accuracy:** Plot accuracy over time (e.g., per hour or day) to see if the model is stable or experiences drift.
- **Probability Calibration:** Compare predicted probabilities vs. empirical frequencies of events in each class (optional).

9 Results and Discussion

If implemented correctly:

- The **multi-class logistic regression** should provide a probabilistic interpretation of whether a coin's price will go up, remain neutral, or go down.
- When cross-asset features (particularly BTC's order book signals) are included, altcoins (e.g., ETH, BNB) may see improved predictive accuracy due to BTC's influence on general market sentiment.

In practice, exact performance will depend on:

1. Volume of data and sampling frequency.
2. Quality of feature engineering (e.g., correct alignment, carefully chosen rolling windows).
3. Choice of α for label thresholding.
4. Hyperparameter tuning (e.g., learning rate α) and regularization (if used).

10 Conclusion

This project demonstrates an end-to-end **supervised learning pipeline** for multi-class price movement classification on **high-frequency order book data**. By leveraging multi-class (softmax) logistic regression and carefully constructed features, we can achieve interpretable predictions without relying on external libraries.

Future extensions might include:

- Exploring *regularization* techniques (L1, L2) to mitigate potential overfitting in high-dimensional data.
- Testing *nonlinear* approaches (e.g., neural networks, decision trees) if permitted in advanced settings.
- Incorporating *additional assets* or alternative data sources (e.g., social media sentiment).

References

[1] Binance Official API Documentation. `https://binance-docs.github.io/apidocs/spot/en/`

[2] Ng, A. (2002). *CS229 Lecture Notes on Logistic Regression*. Stanford University.

[3] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.