Market Regime Detection in Cryptocurrency Markets: A Deep Learning Approach

By Dustin M. Haggett

Abstract

This project presents a comprehensive machine learning approach for classifying market regimes within the BTC/USDT perpetual futures market. I developed a regime-aware model using deep learning techniques, specifically an LSTM architecture enhanced with multi-head attention. The model was trained on a high-frequency dataset comprising over 105,000 samples of 15-minute interval data, incorporating 23 technical indicators that span momentum, trend, volatility, and volume dimensions. To reduce multicollinearity and dimensionality, I performed correlation analysis and applied Principal Component Analysis (PCA), retaining 11 components that capture over 96.6% of the variance. The final model achieved 89.1% accuracy on the test set, with strong performance in bearish (91.6%) and bullish (91.2%) regimes and slightly lower accuracy in neutral regimes (85.0%). These results support the effectiveness of regime-specific feature engineering and attention mechanisms in financial time series classification.

1. Introduction

1.1 Background

Cryptocurrency markets exhibit distinct behavioral regimes, commonly classified as bearish, bullish, or neutral. These regimes reflect shifts in price dynamics, trading volume, volatility, and market sentiment. Real-time detection of such regimes is essential for deploying adaptive trading strategies, particularly in high-leverage markets like BTC/USDT perpetual futures. However, identifying these regimes is challenging due to the markets' volatile and non-stationary nature.

1.2 Objectives

My main objectives in this project were to:

(1) design a robust classification model to identify market regimes based on historical technical indicators;

(2) reduce input dimensionality and address multicollinearity;

(3) improve classification performance through the use of deep learning and attention mechanisms; and

(4) interpret the results in the context of practical algorithmic trading.

2. Data Preprocessing and Feature Engineering

2.1 Dataset Overview

I collected 105,380 samples of 15-minute BTC/USDT perpetual futures data using Binance's API. Each data point includes OHLCV (Open, High, Low, Close, Volume) values and 23 derived technical indicators categorized into the following groups:

- **Trend**: EMA-50, ADX, Trend Strength
- **Momentum**: RSI, ROCR-48, Awesome Oscillator
- **Volatility**: Ulcer Index, Bollinger Band Width
- **Volume**: MFI, CMF, Volume Normalized, EOM
- **Risk**: Funding Volatility, Ulcer Index

2.2 Preprocessing Steps

To prepare the data, I applied the following steps:

- Standardized all features using StandardScaler.
- Performed PCA for dimensionality reduction, retaining 11 components that explained 96.6% of the variance.
- Used a correlation matrix to identify and remove feature pairs with correlation coefficients above 0.7.
- Generated regime labels using smoothed rolling returns over multiple time horizons.

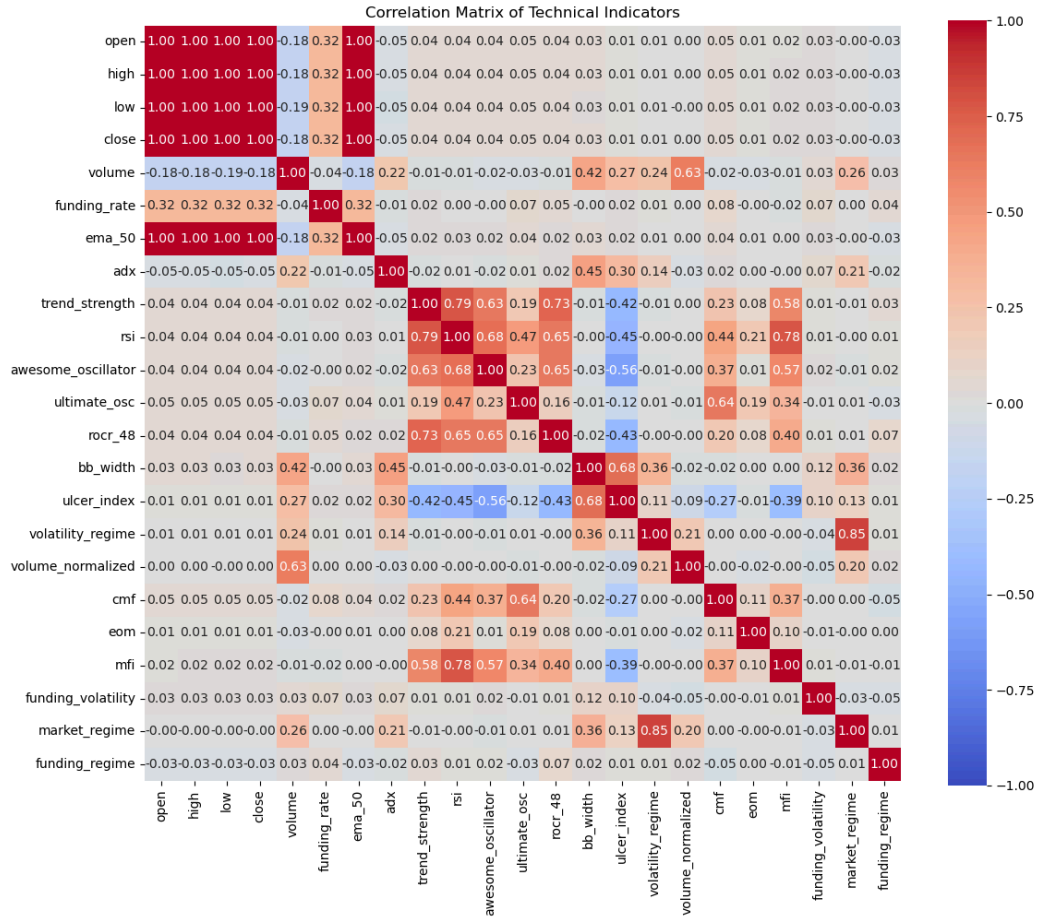

Figure 1. Correlation matrix of all technical indicators. Indicators with a pairwise correlation above 0.7 were flagged for removal to reduce multicollinearity and avoid redundancy in the input space.

2.3 Feature Selection

I assessed feature importance through PCA loadings, Random Forest feature scores, and Mutual Information analysis. Indicators such as Volume, Bollinger Band Width, ROCR-48, Ulcer Index, and EMA-50 proved to be the most predictive across different regimes. Additionally, I implemented regime-specific feature selection to reflect how feature relevance shifts depending on market conditions.

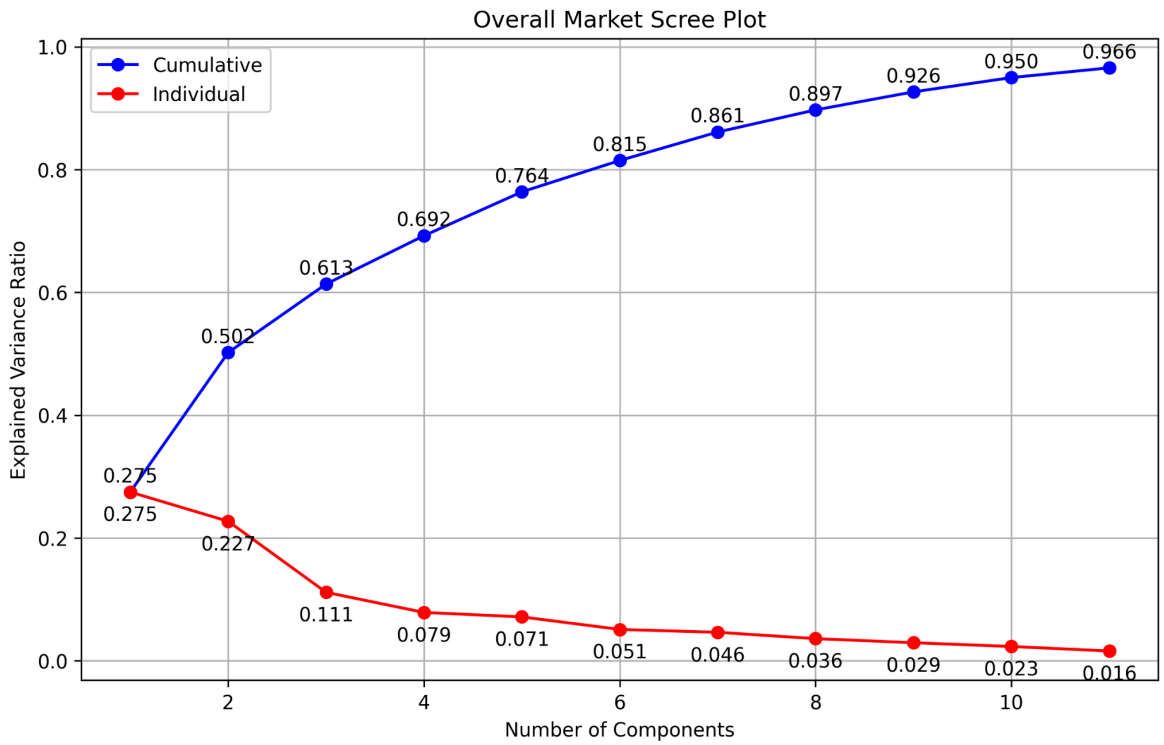

Figure 2. PCA scree plot showing cumulative and individual explained variance ratios. The top 11 components were retained, capturing 96.6% of the total variance.

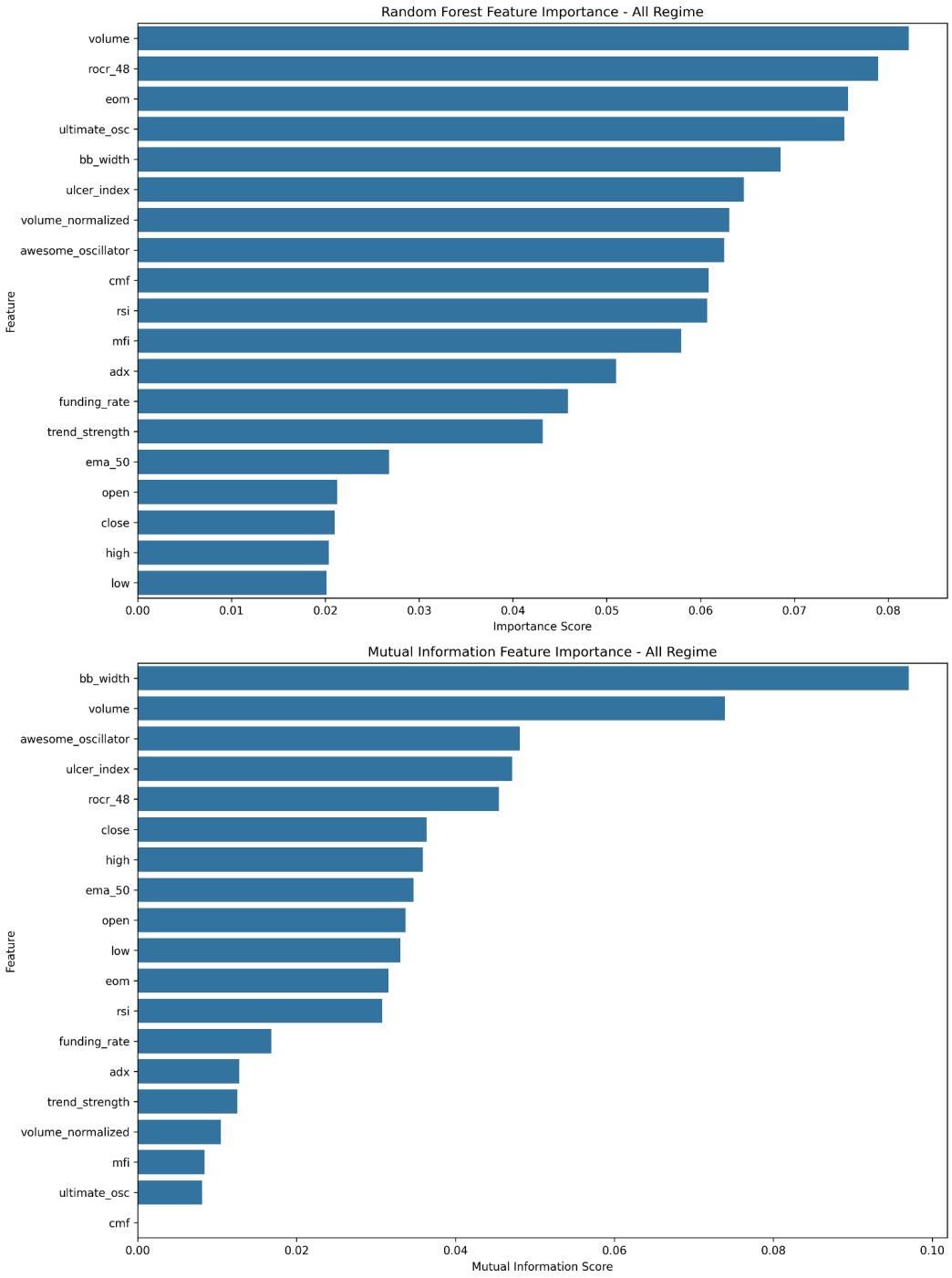

Figure 3. Feature importance scores using Random Forest (top) and Mutual Information (bottom). Volume, ROCR-48, BB Width, and Ulcer Index consistently ranked among the most informative features across regimes.

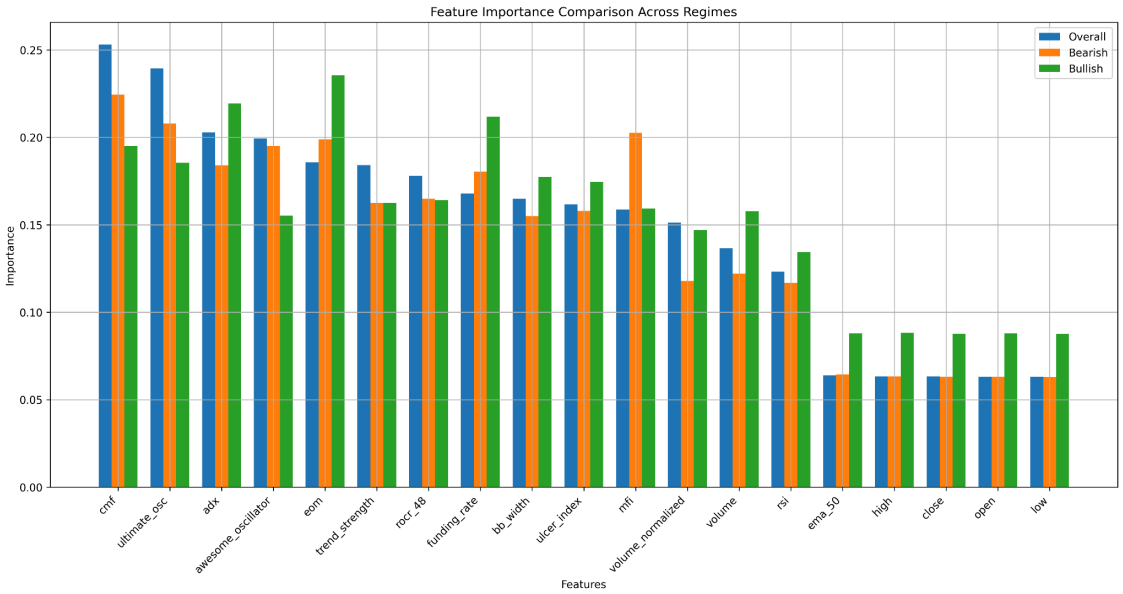

Figure 4. Comparison of feature importance scores across bullish, bearish, and overall market regimes. Notable shifts in feature relevance highlight the benefit of regime-specific feature selection strategies.

3. Modeling Approach

3.1 Architecture

To capture both temporal dependencies and regime-specific patterns, I designed an LSTM-based architecture with the following layers:

- Input normalization tailored per regime
- Two bidirectional LSTM layers (hidden size: 64)
- Multi-head attention mechanism (4 heads)
- Dense output layer with softmax activation for classification

This architecture enabled the model to process sequential data while dynamically focusing on the most relevant features for each regime.

3.2 Training Setup

I trained the model using the AdamW optimizer with a learning rate of 0.001. To address class imbalance, I applied Focal Loss. Early stopping was employed after 10 epochs without improvement on the validation set. I split the dataset into 80% training, 10% validation, and 10% test data.

4. Evaluation and Results

4.1 Performance Metrics

The model achieved the following classification accuracies on the test set:

- **Overall Accuracy**: 89.10%
- **Bearish Regime**: 91.64%
- **Neutral Regime**: 85.02%
- **Bullish Regime**: 91.17%

Analysis of the confusion matrix showed strong diagonal dominance, particularly for the bearish and bullish classes. Most misclassifications occurred in the neutral class, typically overlapping with adjacent regimes.

4.2 Regime Transition Matrix

The transition matrix revealed that the model tends to favor gradual regime transitions (e.g., Bearish → Neutral) over abrupt changes (e.g., Bearish → Bullish). This behavior aligns with the continuity of market sentiment and confirms the model's sensitivity to transitional dynamics.

4.3 Trading Implications

The classifier provides high-confidence signals during established regimes, supporting both long and short directional trades. During neutral phases or regime shifts, I recommend conservative positioning. I also found that the softmax confidence scores can be leveraged to inform dynamic position sizing.

5. Conclusion and Implementation

This project demonstrates the effectiveness of combining LSTM architectures with attention mechanisms for market regime detection. By incorporating dimensionality reduction and regime-aware feature selection, I significantly improved the model's accuracy and

interpretability. These findings have practical implications for real-time trading strategies in volatile crypto markets.

Future Work

I plan to extend this research in several directions:

- Generalize the model to detect regimes across multiple crypto assets
- Explore transformer-based architectures for enhanced sequence modeling
- Integrate sentiment and news data to enrich the feature space

How to Run and Codebase Overview

This project follows a modular architecture structured around data processing, modeling, evaluation, and deployment. Below is a breakdown of key components and their roles in the workflow.

/code/modeling/

- **lstm_trainer.py**

  Trains a regime-aware LSTM model with multi-head attention. Includes early stopping, class weighting, and temperature calibration.

- **feature_engineering.py**

  Computes technical indicators (trend, momentum, volume, volatility), performs preprocessing, and applies regime labeling.

- **backtesting.py**

  Simulates strategy performance, calculates PnL, Sharpe ratio, drawdown, and visualizes results.

- **walk_forward_test.py**

  Performs rolling-window walk-forward validation with confidence analysis.

- **trading_strategy.py**

  Generates trade signals and manages risk via regime-specific position sizing logic.

- **utils.py**

  Shared helper functions for metrics, processing, and plotting.

/code/analysis/

- **data_diagnostic.py**

  Runs quality checks, visualizes feature distributions, and analyzes correlation and regime prevalence.

- **forward_test.py**

  Evaluates model performance on out-of-sample data, generating final performance reports.

- **retrain_model.py**

  Automates model retraining with version tracking.

- **update_data.py**

  Manages market data updates and synchronization across the pipeline.

/code/data_pipeline/

- **data_collector.py**

  Interfaces with the Binance API, performs data cleaning, and stores processed market data.

/models/lstm/

- **regime_lstm_calibrated.pth**

  Calibrated model weights.

- **scaler.pkl**

  Saved StandardScaler object for consistent input normalization.

- **training_history_calibrated.json**

  Training logs for post-hoc evaluation.

- **feature_names.txt**

  Feature list used at train time to ensure compatibility.

/data/processed/

- **processed_data.csv**

  Cleaned and labeled input data for training.

- **walk_forward_data.csv**

  Structured splits for walk-forward testing.

environment.yml

Defines the full Conda environment, including PyTorch, NumPy, pandas, TA-lib, and matplotlib. Run conda env create -f environment.yml to replicate the development setup.

Execution Order

1. **Run data_collector.py** to collect the most recent market data
2. **Run feature_engineering.py** to compute indicators and generate labels
3. **Run lstm_trainer.py** to train and calibrate the model
4. **Use backtesting.py or walk_forward_test.py** for performance validation
5. **Deploy with trading_strategy.py** to simulate real-time usage