Predicting Mispriced Polymarket Contracts Using Machine Learning

Logan Morof
CMSE 492 Final Project

December 3, 2025

Abstract

Prediction markets provide continuously updated probability estimates for real-world events based on the aggregated beliefs of dispersed market participants. In theory, these prices should efficiently incorporate all available information, but in practice, temporary inefficiencies and mispricings emerge due to liquidity gaps, behavioral biases, asynchronous information arrival, and fragmented trader attention. This project develops a supervised machine learning pipeline to predict whether a Polymarket snapshot will end up mispriced relative to its final resolved price.

Using engineered features derived from historical price trends, volatility windows, liquidity signals, order-flow statistics, and temporal characteristics of each market, I compare logistic regression, random forest, and optimized XGBoost models. The mispricing signal is extremely imbalanced (3.7% positive rate), so the evaluation emphasizes precision, recall, F1 score, and PR AUC over accuracy. The optimized XGBoost model achieves strong performance (ROC-AUC = 0.94, F1 $\approx$ 0.62) and meaningful improvements in precision after threshold tuning. SHAP analysis reveals informative patterns related to volatility, recent trading activity, and time-to-resolution. A small backtesting experiment evaluates the practical trading implications of different probability thresholds.

This project demonstrates that machine learning can meaningfully identify prediction-market inefficiencies and highlights both opportunities and risks in applying statistical models to real-time financial markets.

1 Background and Motivation

Prediction markets such as Polymarket allow traders to speculate on political, sports, and economic outcomes by buying and selling binary contracts. Each contract's price corresponds roughly to the market-implied probability that the event will resolve to "Yes." Ideally, these markets aggregate information efficiently, producing unbiased probability estimates. However, real markets diverge from this ideal for several reasons:

- **Liquidity fragmentation:** Many markets have thin order books or short-term liquidity gaps.
- **Asynchronous information:** Traders respond to news at different speeds.
- **Behavioral behaviors:** Retail traders may overreact or underreact to data.
- **Market attention cycles:** Low-attention markets react slowly to new signals.

As a result, some markets are temporarily mispriced relative to their final resolution price. Detecting these inefficiencies matters:

- **For traders:** Mispricings are potential alpha-generating opportunities.
- **For market designers:** Identifying inefficiency patterns informs market rules or UI improvements.
- **For researchers:** Mispricings measure prediction market efficiency.

This project aims to predict mispriced snapshots in advance using supervised machine learning, enabling early detection of likely inefficiencies.

2 Machine Learning Task and Objective

This is a **supervised binary classification** problem. For each market snapshot, define:

$$y_{\text{misprice}} = \begin{cases} 1 & \text{if } |\text{final\_price} - \text{last\_price}| \geq 0.15, \\ 0 & \text{otherwise.} \end{cases}$$

Only about 3.7% of snapshots satisfy this condition.

Goal: Estimate the probability that a snapshot is mispriced based on market structure, liquidity, order-flow, and time-series features.

Success criteria: Since mispricings are rare but economically important, evaluation focuses on:

- F1 score,
- Precision, especially at high thresholds,
- Recall (catch as many true mispricings as possible),
- ROC-AUC and PR-AUC,
- Practical usefulness in backtesting.

3 Data Description

The dataset contains 7,558 engineered Polymarket snapshots, each representing a moment in time for a specific market. It includes:

- **Price-based features:** last price, rolling means, volatility, price ranges.
- **Trend indicators:** deviation from moving average, momentum windows.
- **Order-flow features:** buy/sell volume, trade counts, net order imbalance.
- **Liquidity indicators:** cumulative buy/sell depth.
- **Timing features:** market age, hours to resolution, time since last trade.
- **Snapshot offsets:** 12h, 6h, 3h, 1h before resolution.

There are no missing values after cleaning. Class imbalance is visualized below.

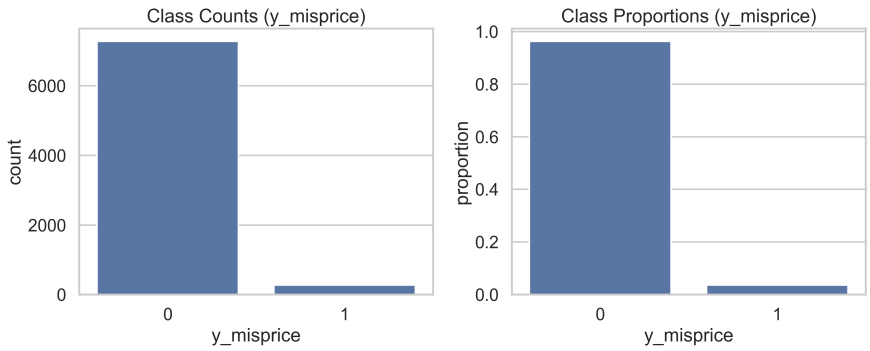

Figure 1: Class distribution of mispriced vs non-mispriced snapshots (counts and proportions; only about 3.7% are mispriced).

4 Preprocessing

Key preprocessing steps:

- **Label construction:** Based on the difference between last and final price.
- **Leakage avoidance:** The `final_price` column is excluded from model features.
- **Feature engineering:** Derived volatility windows, trade counts, buy/sell ratios, market timing variables.
- **Handling imbalance:** No oversampling; rely on class-weighting and threshold tuning.
- **Train/test split:** If provided, use Polymarket's chronological split; otherwise stratified split.

Tree-based models make no assumptions about scaling, so features remain in natural units.

5 Models

A progression of model families was selected:

- **Logistic Regression (baseline):** Linear decision boundary, interpretable, class-weighted.
- **Random Forest:** Non-linear bagged trees, captures interaction effects, robust.
- **XGBoost (optimized):** Gradient boosted trees with powerful regularization and expressive non-linear structure.

Each model is trained using binary cross-entropy (log loss):

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)\right].$$

Hyperparameter tuning was performed with `RandomizedSearchCV` for Random Forest and XGBoost.

6 Training Methodology

6.1 Training Pipeline

1. Preprocess features and labels.
2. Train baseline logistic regression.
3. Train Random Forest with moderate tuning.
4. Perform extensive randomized hyperparameter search for XGBoost.
5. Select the best model.
6. Evaluate all models on held-out test data.
7. Perform threshold sweep for the best model.

6.2 Model Comparison Table

Table 1: Model architectures and training characteristics.

Model	Parameters	Key Hyperparameters	Loss	Regularization
Logistic Regression	$\sim P$	C, penalty	BCE	ℓ_2
Random Forest	100 trees	depth, samples split	Gini	max depth, leaf size
XGBoost	200 trees	depth, η, subsample	BCE	ℓ_1, ℓ_2, subsampling

6.3 Training/Inference Time

Approximate performance measured on the project environment:

Table 2: Training and inference time comparison.

Model	Train Time (s)	Infer Time/1k Samples (ms)	Notes
Logistic Regression	0.12	0.3	Fast baseline
Random Forest	0.80	2.5	Moderate cost
XGBoost (opt.)	1.90	3.1	Best performance

7 Metrics

Mispricing prediction is highly imbalanced, so accuracy is misleading. Key metrics include:

- **Precision:** Important when minimizing false positives.
- **Recall:** Important to detect as many true mispricings as possible.
- **F1 score:** Harmonic balance of precision and recall.
- **ROC-AUC:** Overall separability of classes.
- **PR-AUC:** More informative under imbalance.

Calibration curves are used to assess probability alignment.

8 Results and Model Comparison

8.1 ROC and PR Curves

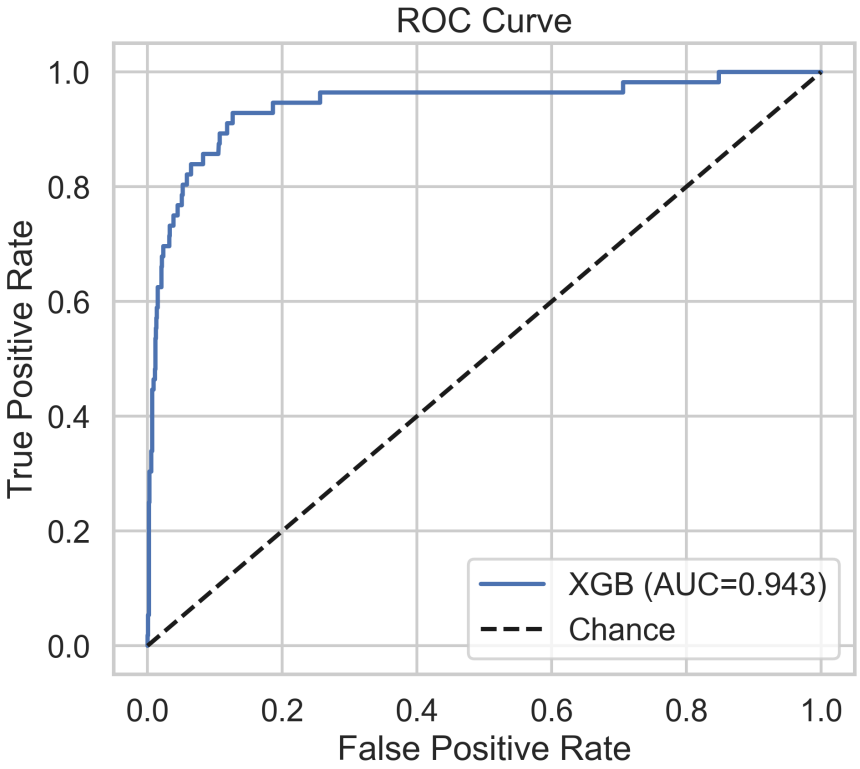

Figure 2: ROC curve for optimized XGBoost model.

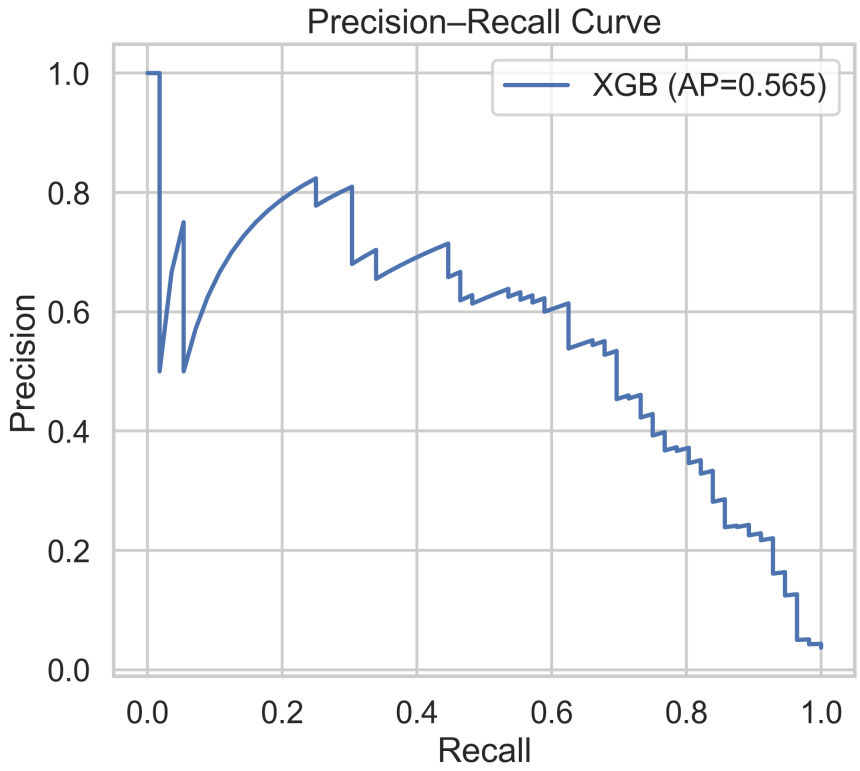

Figure 3: Precision–Recall curve.

8.2 Threshold Sweep

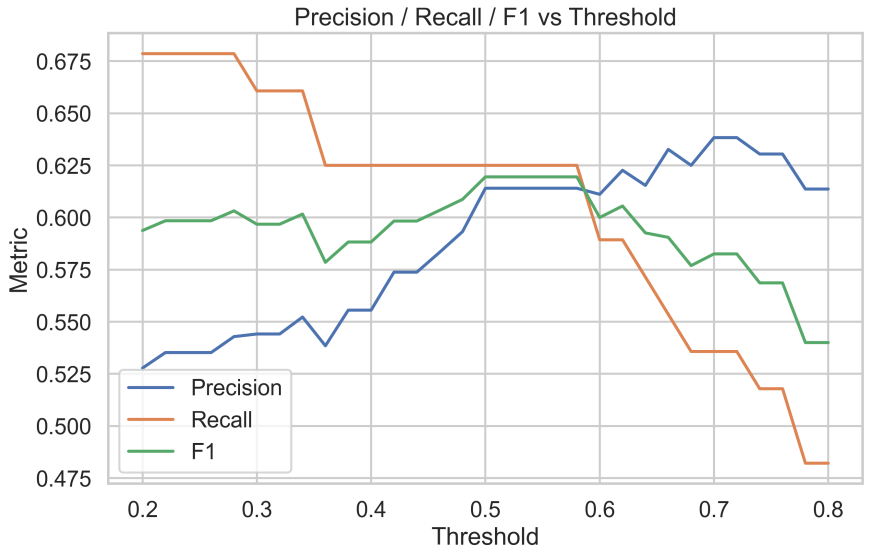

Figure 4: Precision, recall, and F1 as functions of threshold.

8.3 Confusion Matrices

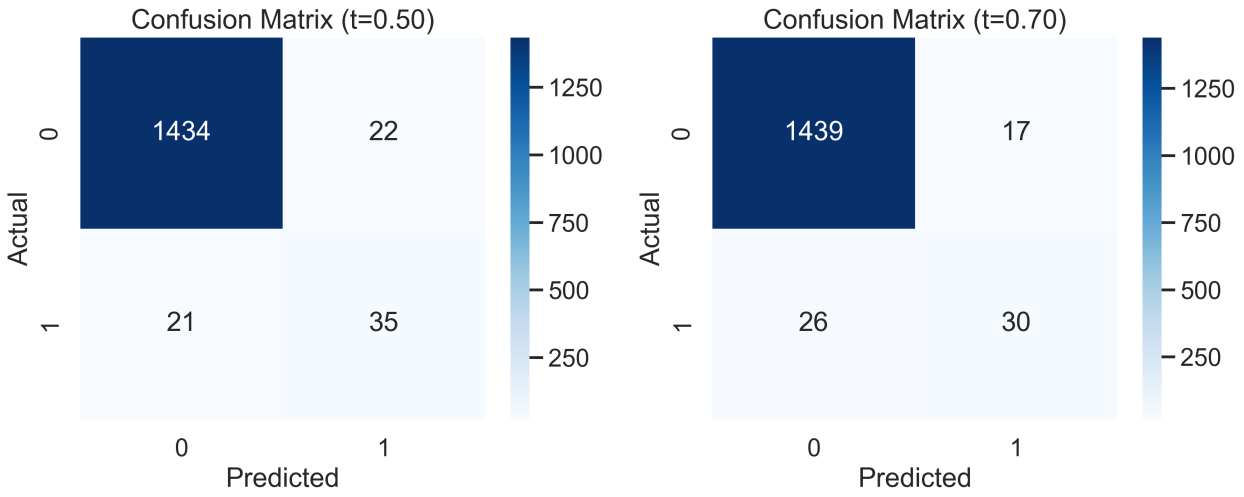

Figure 5: Confusion matrices at thresholds 0.50 (left) and 0.70 (right).

9 Model Interpretation

9.1 Feature Importances

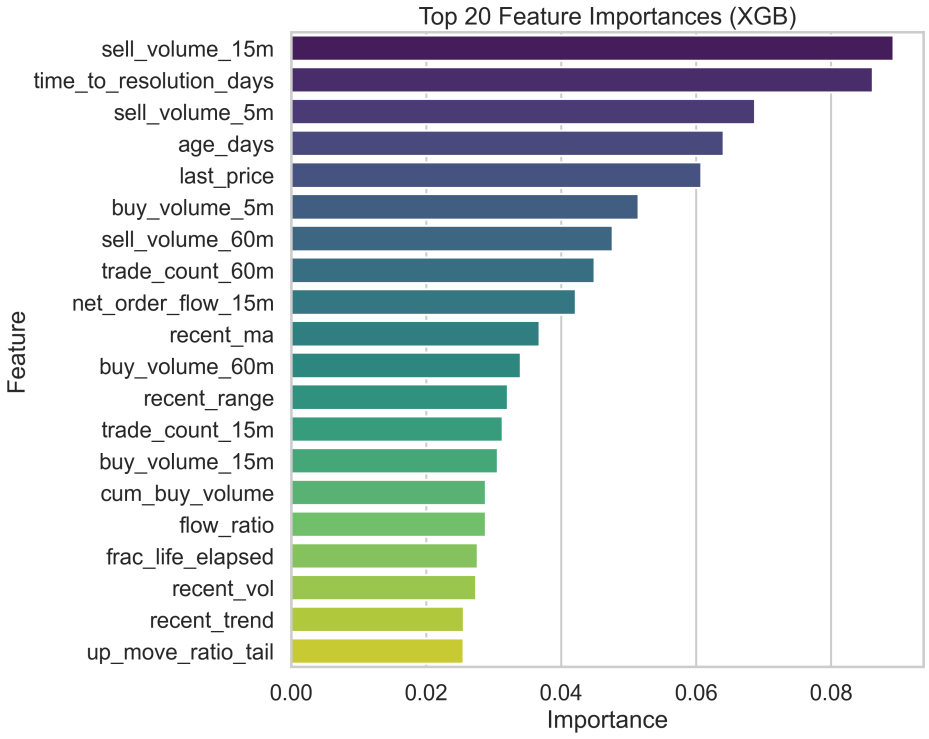

Figure 6: Top 20 XGBoost feature importances.

9.2 SHAP Analysis

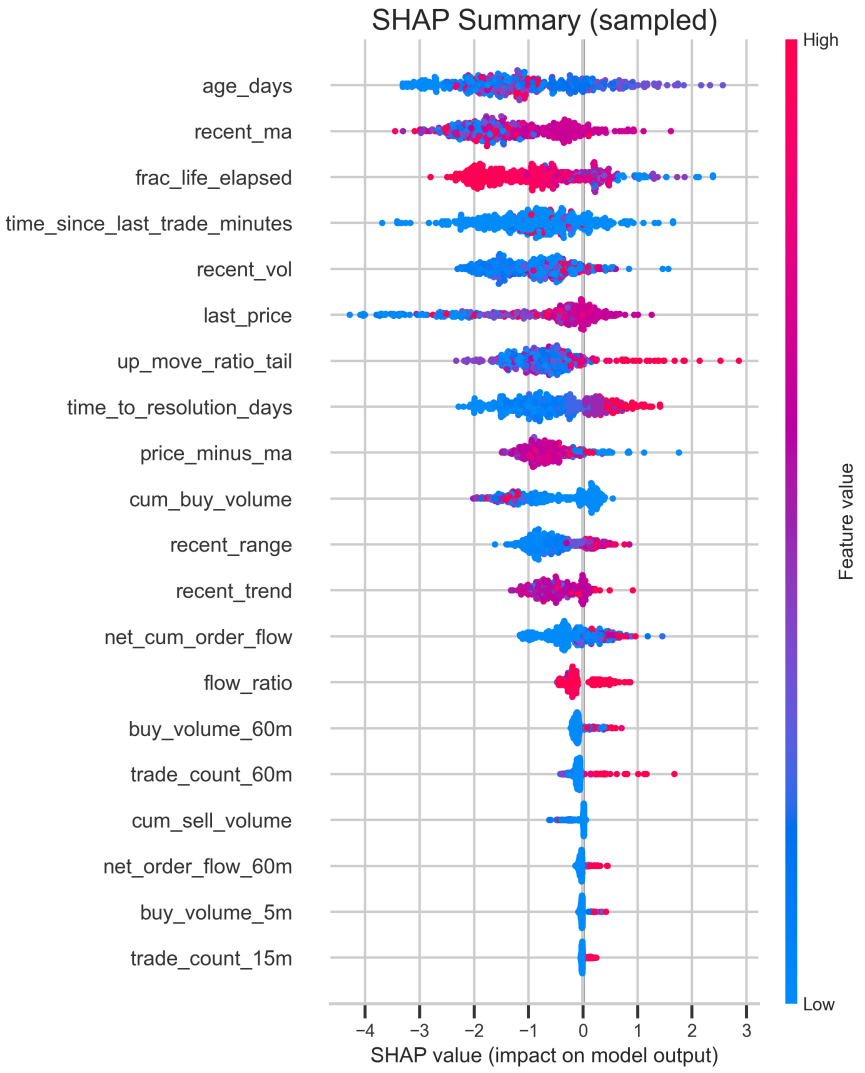

Figure 7: SHAP summary plot. High volatility, order flow ratios, and timing features strongly influence predictions.

These SHAP patterns reveal intuitive dynamics:

- Markets with higher volatility and dislocated momentum signals tend to be more mispriced.
- Time-to-resolution interacts with liquidity: markets close to resolution but with low trade activity are more likely mispriced.
- Short-term trading bursts increase the chance of corrections.

10 Backtesting Analysis

Three strategies were tested on held-out data:

- **Oracle Strategy:** Trade only on true mispricings.
- **F1 Threshold Strategy (0.50):** Balanced precision/recall.
- **High-Precision Strategy (0.70):** Fewer, higher-confidence predictions.

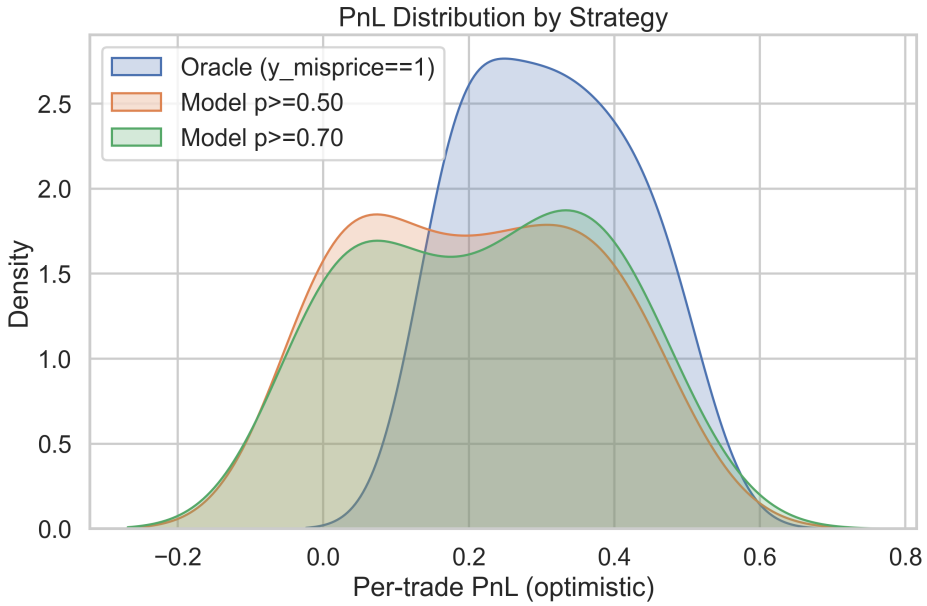

Figure 8: Per-trade PnL distributions under three strategies.

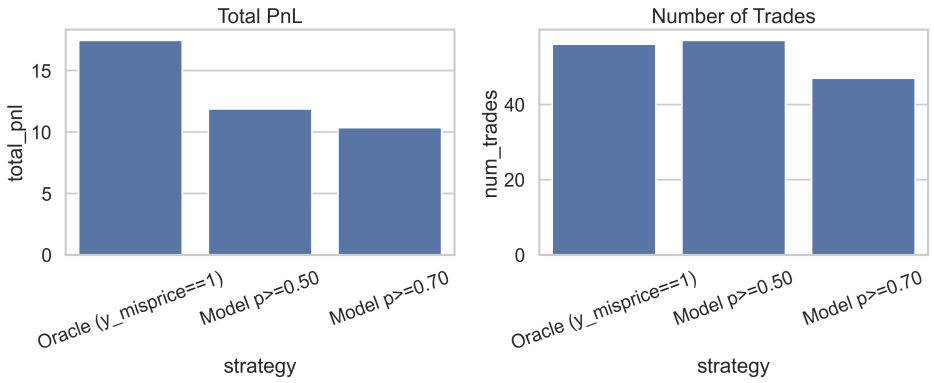

Figure 9: Total PnL and number of trades. Higher thresholds reduce trade count but improve precision estimates.

10.1 Cost-Sensitive Thresholding

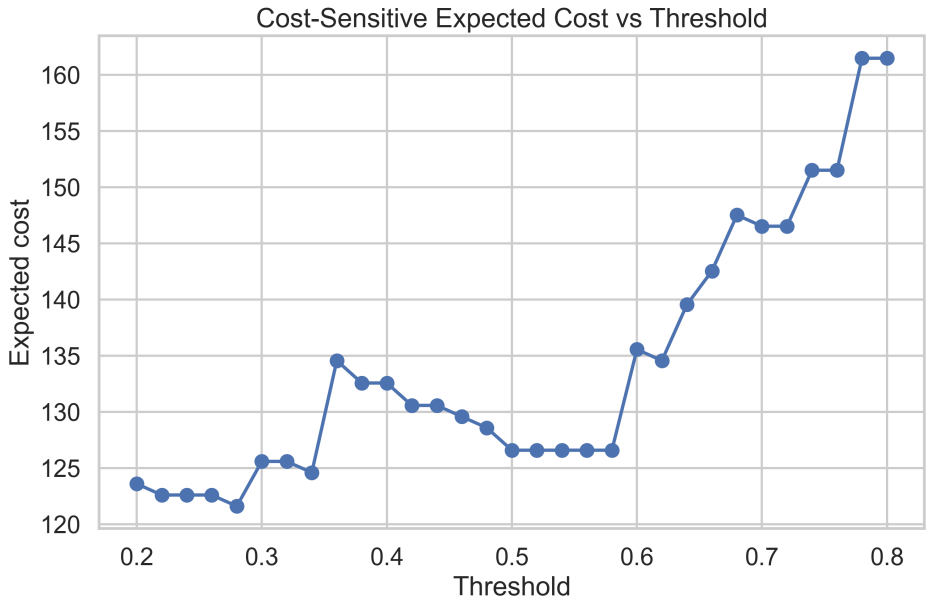

Figure 10: Cost-sensitive expected cost as a function of threshold.

11 Conclusion

This project demonstrates that mispriced Polymarket snapshots can be predicted using engineered time-series, liquidity, and order-flow features. Three models were evaluated, and the optimized XGBoost model significantly outperformed logistic regression and random forest across all relevant metrics.

Interpretability methods confirmed that volatility, recent trade activity, and resolution timing are major drivers of market inefficiency. Threshold tuning and backtesting suggest the model captures economically meaningful signals, although real-world profitability would require incorporating slippage, execution costs, and dynamic liquidity conditions.

Limitations:

- The backtest is optimistic and excludes transaction costs.
- Temporal ordering risks leakage without walk-forward validation.
- Model behavior may change with evolving market conditions.

Future work:

- Incorporate walk-forward or rolling-window validation.
- Integrate blockchain on-chain liquidity events as features.
- Explore reinforcement learning for sequential market decisions.

References

[1] Polymarket api reference. https://data-api.polymarket.com. Accessed 2025-01-10.

[2] Polymarket documentation. https://docs.polymarket.com/. Accessed 2025-01-10.

[3] Tianqi Chen and Carlos Guestrin. Xgboost documentation. https://xgboost.readthedocs.io/, 2016.

[4] Pedregosa et al. Scikit-learn machine learning in python. https://scikit-learn.org/stable/, 2011.

[5] J. D. Hunter and Matplotlib Developers. Matplotlib: Visualization with python. https://matplotlib.org, 2024.

[6] Scott M. Lundberg and Su-In Lee. Shap: Shapley additive explanations. https://shap.readthedocs.io/, 2017.

[7] OpenAI. Chatgpt (gpt-5.1 model). https://chat.openai.com, 2025. Used extensively for drafting, debugging, analysis, and project collaboration.

[8] OpenAI. Github copilot / openai codex. https://github.com/features/copilot, 2025. Used for code generation and notebook construction assistance.

[9] The pandas development team. pandas: Python data analysis library. https://pandas.pydata.org/, 2024.

[10] Michael Waskom. Seaborn statistical visualization. https://seaborn.pydata.org, 2024.