

Predicting Mispriced Polymarket Contracts Using Machine Learning

Logan Morof
CMSE 492 Final Project

December 7, 2025

Abstract

Prediction markets are designed to aggregate information into prices that reflect the probability of real-world events, but in practice those prices wobble because liquidity is uneven, news is absorbed at different speeds, and traders behave imperfectly. In this project I build a supervised machine learning pipeline to flag Polymarket snapshots that later resolve far from their last traded price. I start with a small set of resolved markets to prove out feature engineering, then expand the dataset as the pipeline stabilizes. The task is to detect a mispricing label defined as a 15-cent gap between the final resolution price and the last observed price. Because only 3.7% of snapshots are mispriced, evaluation focuses on precision, recall, F1, PR-AUC, calibration, and threshold behavior rather than accuracy. A tuned XGBoost model achieves the best balance (ROC-AUC of about 0.94, F1 around 0.59 at a 0.50 threshold). SHAP interpretation shows that volatility, order-flow pressure, and time to resolution are key drivers, and a simple backtest illustrates how different probability thresholds trade off coverage versus confidence. The results suggest that machine learning can surface short-lived inefficiencies, while also highlighting the practical hurdles of deploying such signals in live markets.

1 Background and Motivation

Polymarket lets traders buy YES/NO contracts on political, economic, and cultural events. In an efficient world, the price of a YES contract would track the probability of a “Yes” outcome. Real markets drift away from that ideal. Liquidity is lumpy and shallow in many contracts, news flows unevenly to different participants, and attention waxes and wanes. During those gaps, prices can be wrong for hours or days.

Human intuition alone struggles to spot rare, transient mispricings consistently, especially under heavy class imbalance and across many markets at once. A probabilistic model can impose a consistent scoring rule, monitor thousands of contracts simultaneously, and surface weak signals that are easy to overlook.

I initially treated this as an outcome prediction problem—guessing which side would win—but that framing felt unsatisfying for trading: many outcomes are obvious, class imbalance is extreme, and a small edge on an already-correct price is hard to monetize. After early tests, I pivoted to detecting mispricings near resolution: snapshots where the last traded price disagrees by at least fifteen cents with the eventual settlement. That reframing keeps the focus on short-lived, economically meaningful inefficiencies that a trader might plausibly exploit.

2 Machine Learning Task and Objective

The task is binary classification on market snapshots. A snapshot is labeled mispriced if the final resolution price diverges from the last observed price by at least fifteen cents:

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

Only 3.7% of snapshots satisfy this criterion, so the positives are rare. The goal is to produce calibrated probabilities that separate mispricings from the dominant normal cases and to understand how different thresholds translate into different trading styles.

Because accuracy rewards trivial “always no” predictions, I rely on precision, recall, F1, ROC-AUC, PR-AUC, and calibration, plus an explicit look at threshold choices. In other words, I care about ranking and probability quality, not just how often the model guesses the correct class.

3 Data Collection and Description

All data come from Polymarket’s public Gamma, CLOB price history, and data APIs. I began with a small set of resolved markets to prototype feature engineering and model training, then expanded the universe once the pipeline was stable, pulling more price histories and trades. The underlying data therefore reflect real trading behavior, with all of its irregularities and bursts of activity.

Each of the 7,558 snapshots captures a market at a specific time offset before resolution (12h, 6h, 3h, or 1h), combining price, order-flow, and timing context. In total there are 34 numeric features spanning rolling averages, moving volatility, price ranges, deviations from trend, buy/sell volumes, net order flow, trade counts, and timing variables such as age and time to resolution.

After cleaning, there are no missing values (see `missingness_summary.csv`). The absence of missingness follows from how the pipeline is built: every feature is computed directly from raw trade and order-book data for a given snapshot. If a feature cannot be computed (for example, because there were no trades in a window), that snapshot is dropped during feature construction rather than imputed later, so the final modeling table is complete.

The class balance is extremely skewed—279 positives and 7,279 negatives—as shown in the updated class balance plots. This is expected: large, obvious mispricings are rare compared to ordinary, mostly fair prices.

Univariate and bivariate plots reveal several patterns. Many trade-count features (e.g., `trades_15m`) are nearly always zero, highlighting sparse activity in short windows. Volatility and price features (`recent_vol`, `last_price`, `recent_ma`) are long-tailed and often multimodal. Mispriced cases cluster in higher-volatility regimes and at price levels that later move substantially. Scatter plots of last versus final price and of time to resolution versus the absolute price gap show mispriced points where gaps are larger. A correlation heatmap (`correlation_heatmap.png`, `correlation_matrix.csv`) summarizes relationships among numeric features and confirms that many of the rolling-window statistics co-move as expected.

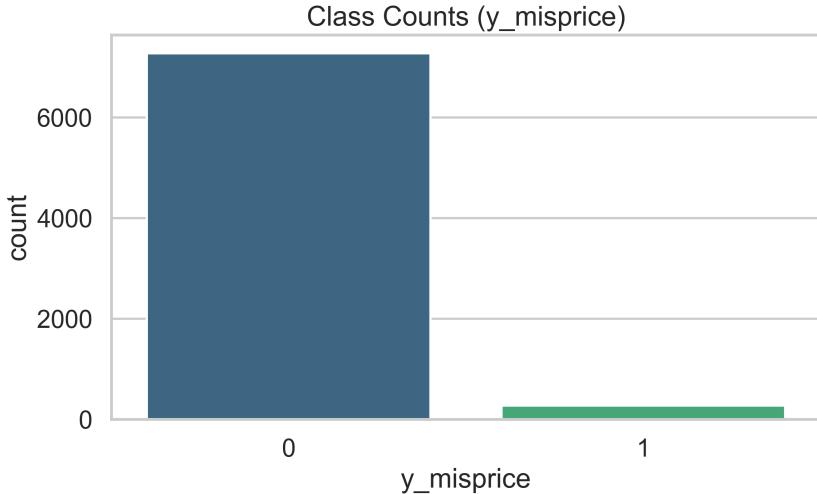


Figure 1: Class distribution of mispriced versus non-mispriced snapshots; only about 3.7% are labeled mispriced.

4 Preprocessing

I recompute y_{misprice} from the final and last prices for reproducibility, so anyone with the same raw data can reconstruct the label. To avoid leakage, I drop `final_price` and any derived price gap used only for plotting before training. Features remain in their natural units; tree-based models do not need scaling, and keeping raw units makes interpretation easier.

The train/test split is an 80/20 stratified split on the mispricing label because no reliable chronological split is available from the API snapshots. Stratification preserves the rare positive rate in both sets. Class imbalance is handled with class weights and threshold tuning rather than oversampling: weighting makes the loss function care more about the rare positives during training, and threshold tuning lets me trade off precision and recall for different use cases.

Although some descriptive plots use the full dataset for context, hyperparameter tuning and model selection are performed exclusively on the training split, with the held-out test set used only for final evaluation.

5 Models and Training Methodology

I move from simple to more expressive models. Logistic regression provides a linear, class-weighted baseline. Random forest adds nonlinear interactions via bagged trees. XGBoost pushes further with gradient boosting and tuned hyperparameters to capture subtle patterns in noisy, imbalanced data.

All models are trained to minimize binary cross-entropy (log loss). For a sample with label $y \in \{0, 1\}$ and predicted probability \hat{p} , the loss is

$$\ell(y, \hat{p}) = -[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})],$$

optionally reweighted to account for class imbalance.

Logistic regression needs minimal tuning beyond the regularization strength C and class weights. The random forest uses a reasonably large number of trees and class balancing, with depth and

leaf parameters controlling regularization. XGBoost uses randomized search over depth, learning rate, subsampling, and regularization terms to find a configuration that performs well without overfitting.

Learning curves show that the linear model has low capacity (train/validation F1 around 0.2), while the forest and XGBoost fit the training data almost perfectly (train F1 near 1.0) and gain validation performance as sample size grows, indicating high capacity and some overfitting risk that is partially controlled by regularization and early stopping.

Table 1: Model characteristics.

Model	Key hyperparameters	Loss	Regularization
Logistic Regression	C (class-weighted), lbfgs	BCE	ℓ_2
Random Forest	500 trees, class_weight=balanced	depth=None, Gini	depth/leaf controls
XGBoost (tuned)	600 trees, depth=6, lr=0.05, subsample=0.7, scale_pos_weight	BCE	ℓ_1 , ℓ_2 , subsampling

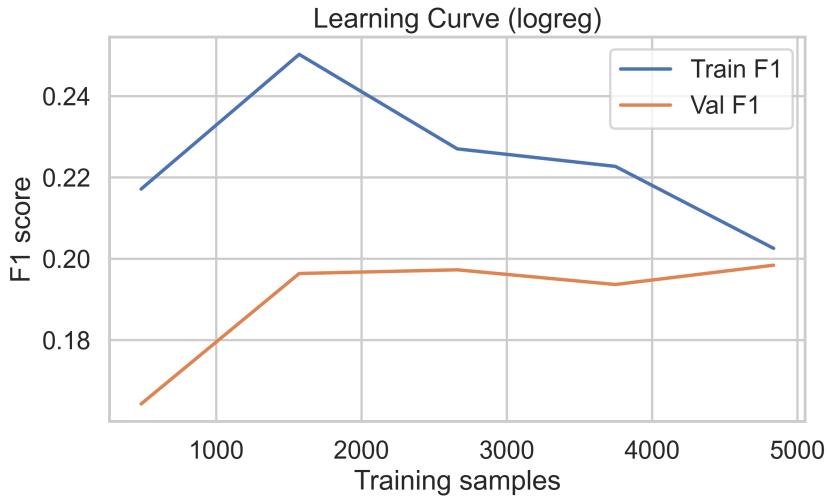


Figure 2: Learning curve for logistic regression. Train and validation F1 hover around 0.2–0.25, reflecting limited capacity for nonlinear structure.

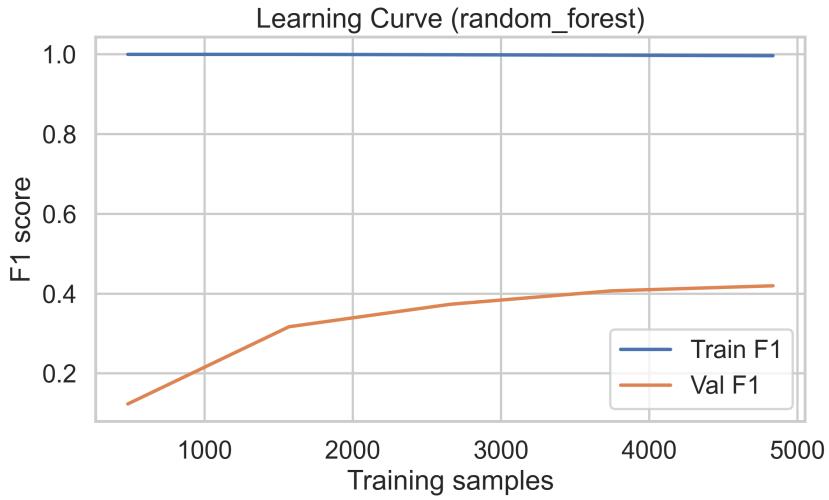


Figure 3: Learning curve for random forest. Train F1 stays near 1.0; validation F1 climbs into the 0.3–0.4 range as sample size grows, showing high capacity and some overfitting risk.

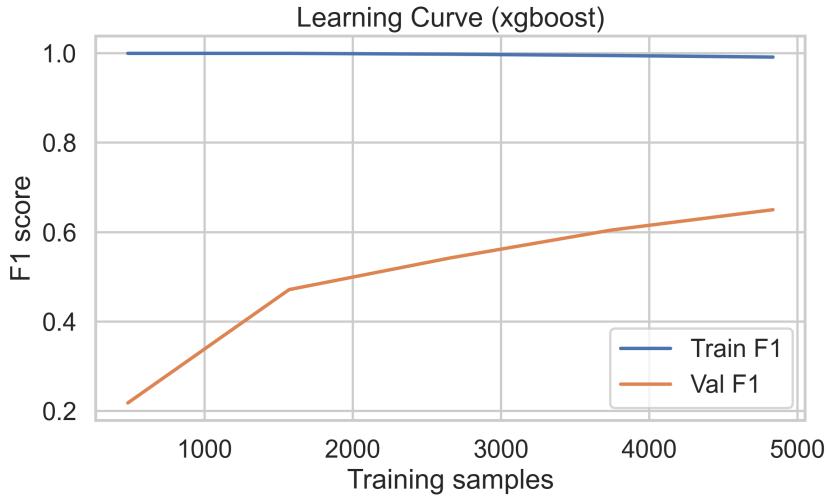


Figure 4: Learning curve for tuned XGBoost. Train F1 is near 1.0; validation F1 rises toward roughly 0.65 with more data, indicating strong capacity with manageable overfit at current sample sizes.

Training and inference remain fast. Logistic regression is effectively instantaneous on this dataset. The random forest and XGBoost models train in about a second and predict in milliseconds per thousand samples, which is easily fast enough for near-real-time scoring.

Table 2: Training and inference time (`model_runtime_comparison.csv`).

Model	Train time (s)	Infer ms / 1k samples	Notes
Logistic Regression	0.042	0.0016	baseline-ish
Random Forest	1.223	0.0531	baseline-ish
XGBoost (tuned)	1.148	0.0089	optimized hyperparams

6 Metrics

With 3.7% positives, accuracy is not a useful metric. A model that always predicts “no mispricing” would achieve high accuracy while providing zero trading value.

Instead, precision answers how many flagged cases are real mispricings; recall measures how many real mispricings are caught; and F1 balances the two. ROC-AUC summarizes ranking quality across all thresholds; PR-AUC focuses on performance in the rare-positive regime; calibration checks how well the predicted probabilities line up with empirical frequencies.

Held-out metrics at the default 0.50 threshold are:

Table 3: Held-out metrics (`model_metrics_comparison.csv`).

Model	Accuracy	Precision	Recall	F1	ROC-AUC	PR-AUC
Logistic Regression	0.7586	0.1089	0.7679	0.1907	0.8383	0.1832
Random Forest	0.9656	0.5667	0.3036	0.3953	0.9433	0.5064
XGBoost (tuned)	0.9689	0.5763	0.6071	0.5913	0.9436	0.5742

Logistic regression achieves respectable recall but very low precision. The tree-based models strike a better balance, with XGBoost delivering the strongest overall metrics.

7 Results and Model Comparison

The ROC curve for tuned XGBoost stays well above the diagonal, and the PR curve stays above the baseline implied by 3.7% positives (average precision about 0.565), indicating good ranking of rare positives.

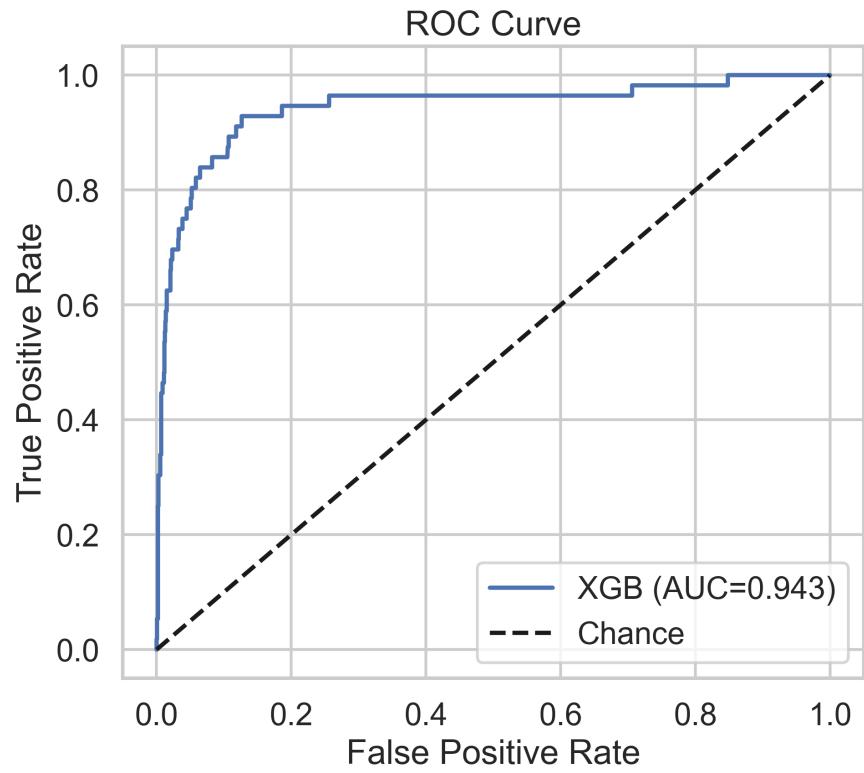


Figure 5: ROC curve for tuned XGBoost; ROC-AUC is about 0.94, meaning true mispricings generally score higher than normal cases.

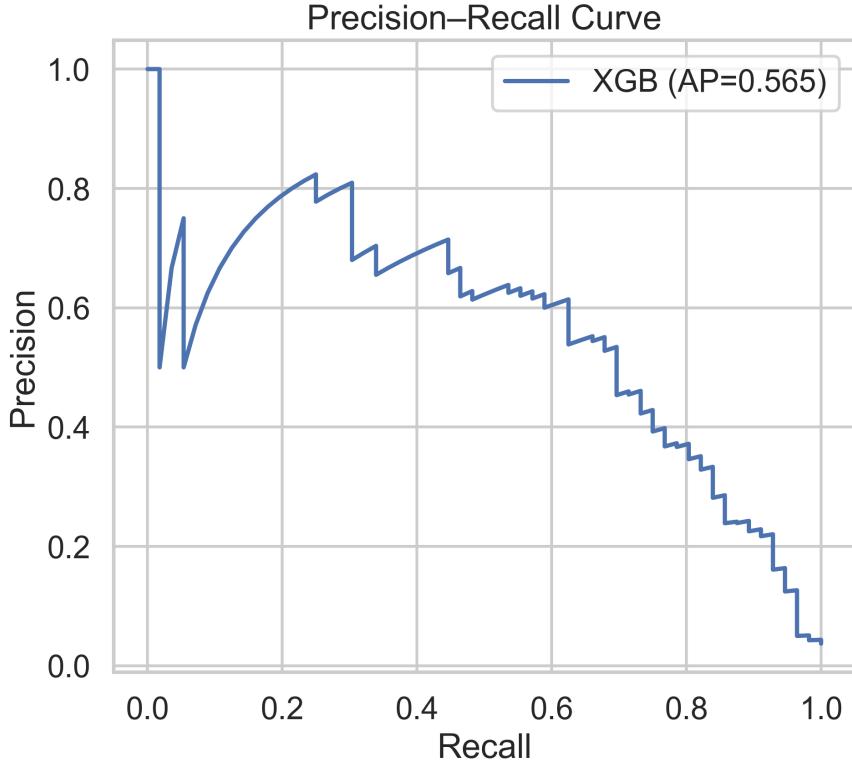


Figure 6: Precision–Recall curve; PR-AUC around 0.565 shows better-than-chance precision even as recall rises under heavy imbalance.

Threshold sweeps show how moving the cutoff trades recall for precision. Around 0.50, F1 peaks; around 0.70, precision improves at the cost of recall. This matches the qualitative sense that more conservative thresholds will miss some opportunities but produce cleaner signals.

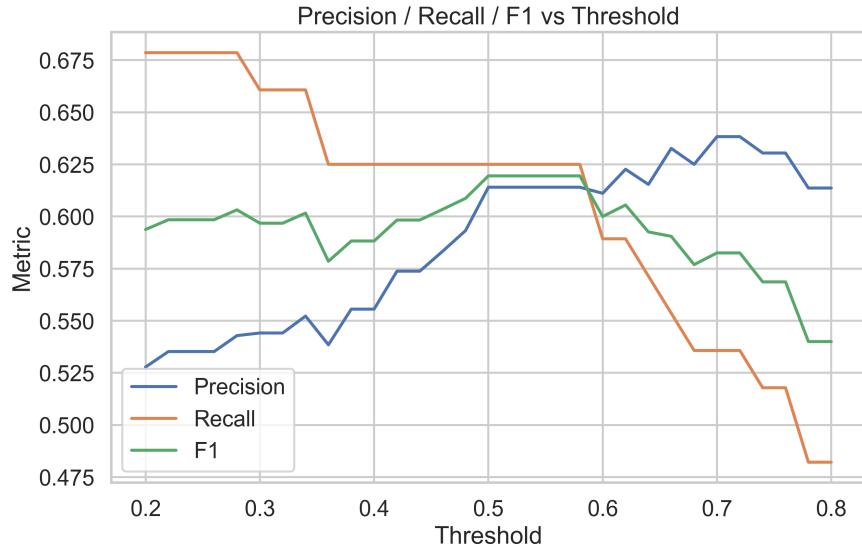


Figure 7: Threshold sweep. Lower thresholds catch more mispricings with more false alarms; higher thresholds are more selective and precise.

Confusion matrices make the trade-offs concrete. At $t = 0.50$, the model finds 35 true mispricings, misses 21, and produces 22 false alarms out of 1,456 normal cases ($TN=1434$, $FP=22$, $FN=21$, $TP=35$). At $t = 0.70$, it finds 30, misses 26, and false alarms drop to 17 ($TN=1439$, $FP=17$, $FN=26$, $TP=30$).

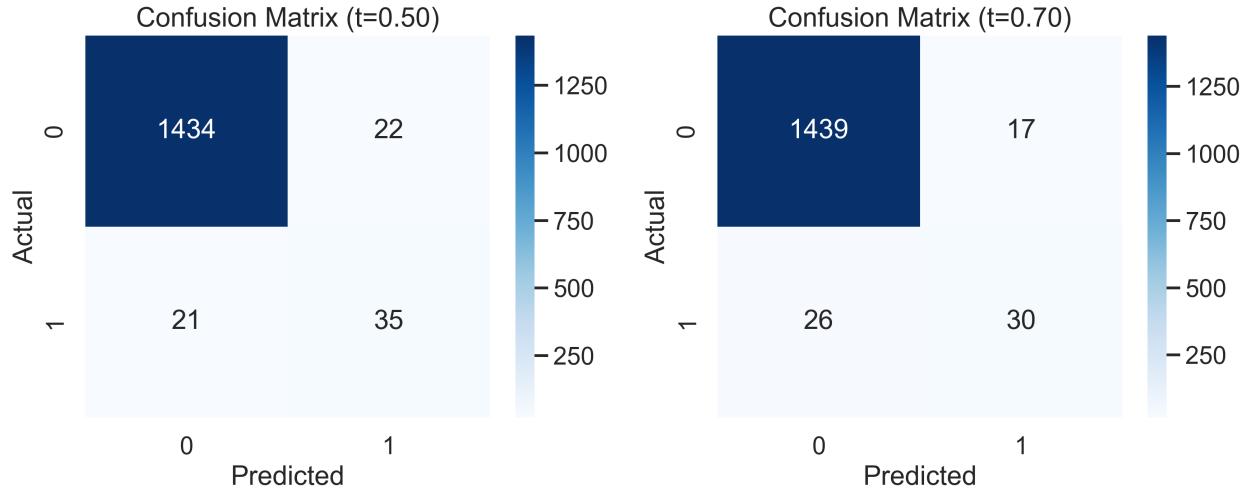


Figure 8: Confusion matrices for $t = 0.50$ (left) and $t = 0.70$ (right). Higher thresholds reduce false positives but miss more true mispricings.

Calibration curves show slight underconfidence in mid-range probabilities: a predicted 40% corresponds to an empirical frequency somewhat higher. That conservatism is acceptable for ranking, but a post-hoc calibration step could help if probabilities directly drive execution or risk sizing.

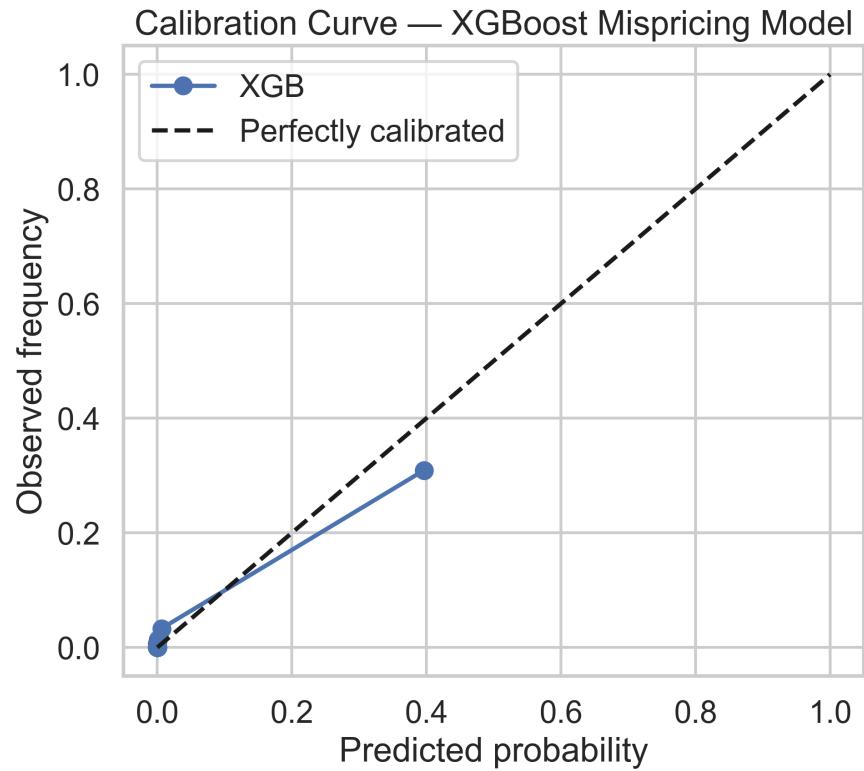


Figure 9: Calibration curve. Mid-range probabilities are conservative; a calibration step could refine them for trading.

A cost-sensitive view echoes the threshold sweep. If missing a mispricing is worse than a false alarm, lower thresholds are favored; if false alarms are costly (for example, because trades move the price or incur fees), higher thresholds are better.

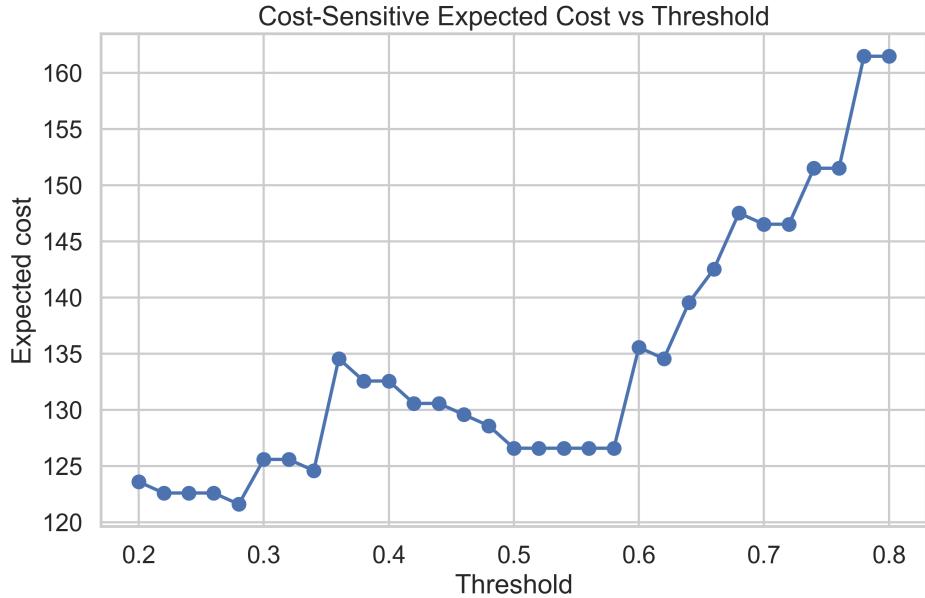


Figure 10: Cost-sensitive expected cost versus threshold. The preferred threshold shifts with the relative cost of misses versus false alarms.

8 Model Interpretation

Feature importances from XGBoost show that short-term volatility, deviations from recent averages, asymmetric order flow, and proximity to resolution dominate the signal. That matches intuition: stressed markets near the finish line are prone to temporary mispricing as traders rush to update positions.

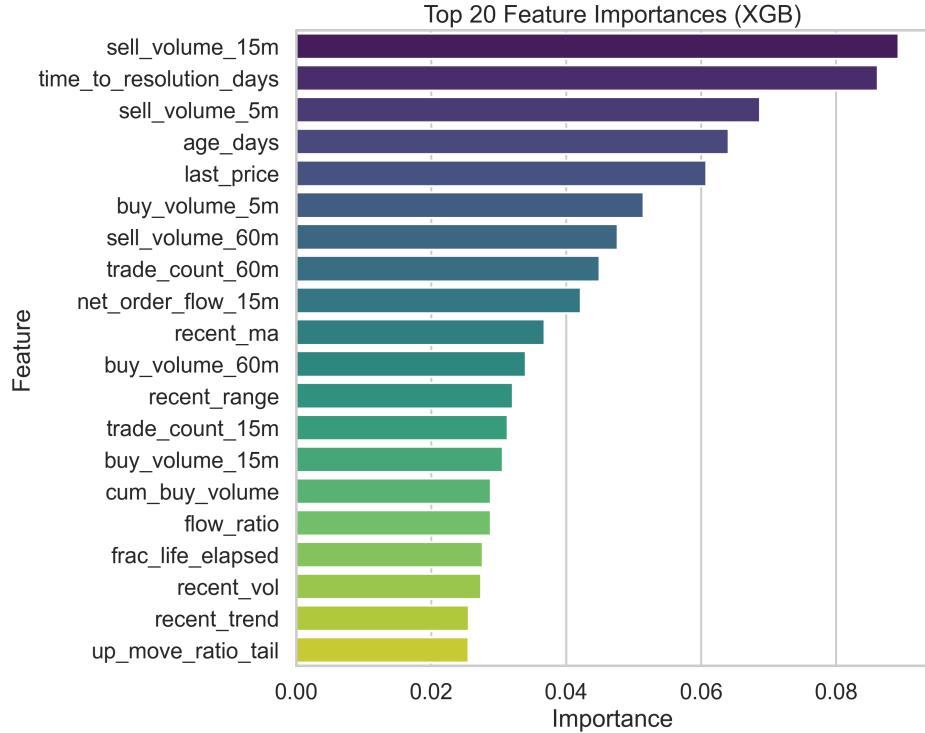


Figure 11: Top XGBoost feature importances. Volatility, order-flow pressure, and time to resolution are key drivers.

SHAP values explain predictions at the instance level. The summary plot shows which features push probabilities up or down across the dataset, while dependence plots reveal how those effects vary with feature values. Heavy recent trading and short time to resolution raise mispricing risk; calm, liquid markets tilt toward fair pricing.

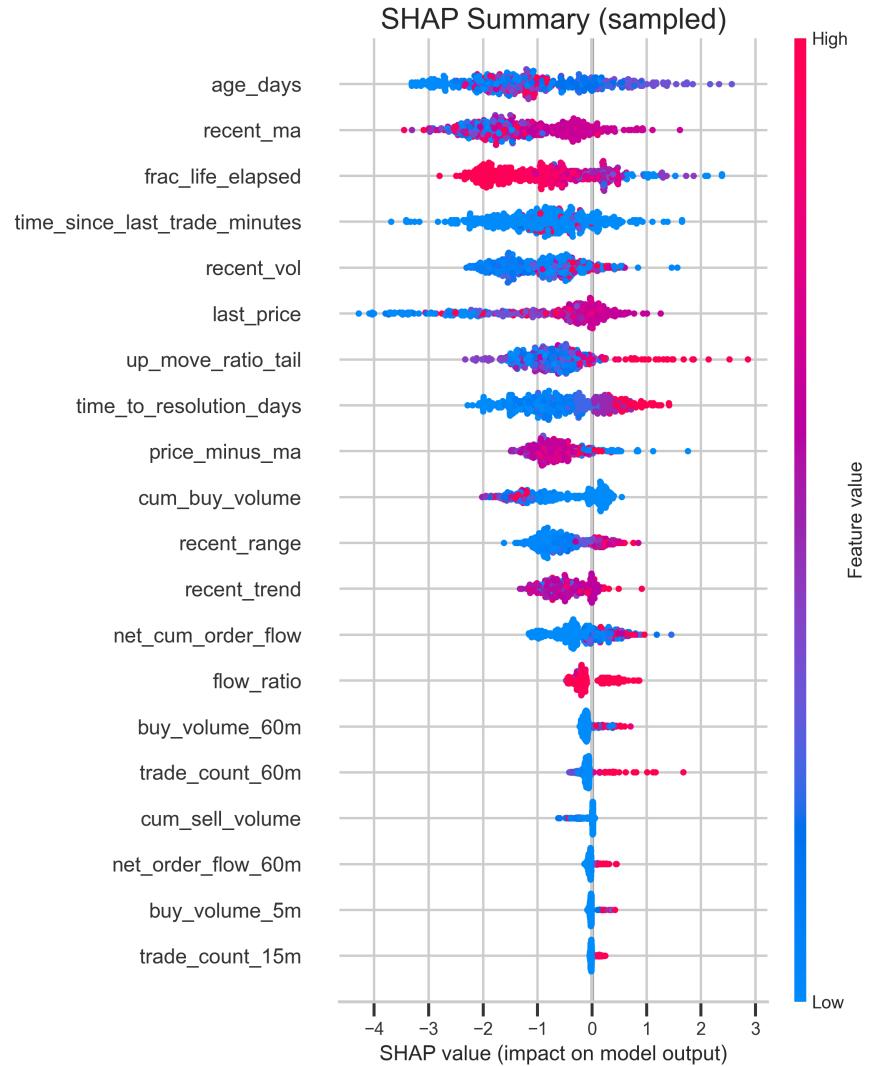


Figure 12: SHAP summary. Wide spreads indicate strong influence; volatility and order-flow asymmetry tend to push predictions toward mispriced.

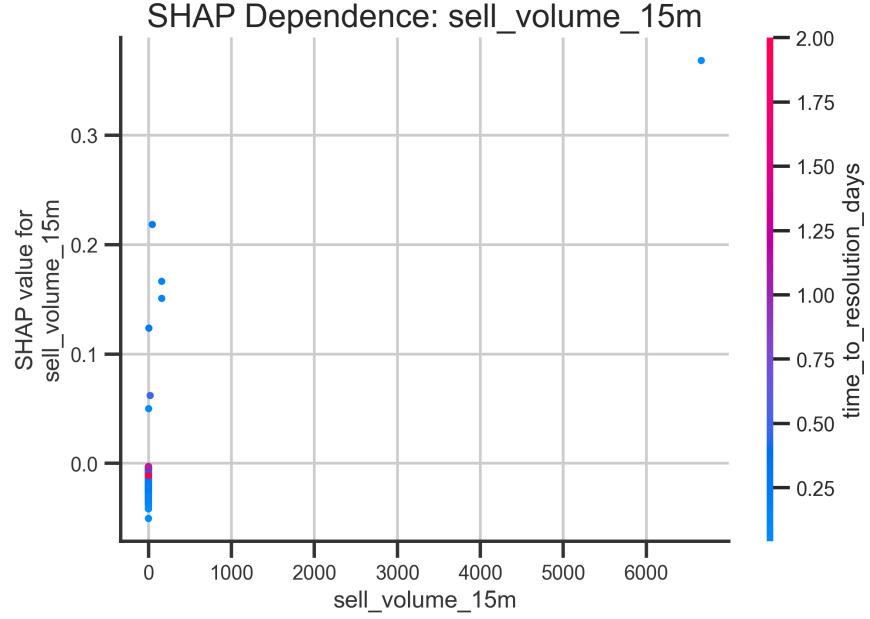


Figure 13: SHAP dependence for fifteen-minute sell volume. Sharp bursts of selling, especially near resolution, increase mispricing risk.

Together, these interpretation tools support the idea that the model is not relying on arbitrary artefacts, but rather on intuitive market signals.

9 Backtesting Analysis

To connect scores to trading implications, I ran a simple optimistic backtest on the held-out set. Per-trade PnL is the absolute gap between final and last price; fees, slippage, position sizing, and order book depth are all ignored. This should be viewed as a best-case scenario for what raw signals could do.

Three strategies are compared: an unattainable oracle that trades only on true mispricings, a balanced 0.50 threshold strategy, and a more selective 0.70 threshold strategy. The oracle earns the most. The 0.50 strategy executes more often with moderate gains; the 0.70 strategy trades less but with higher confidence and steadier outcomes.

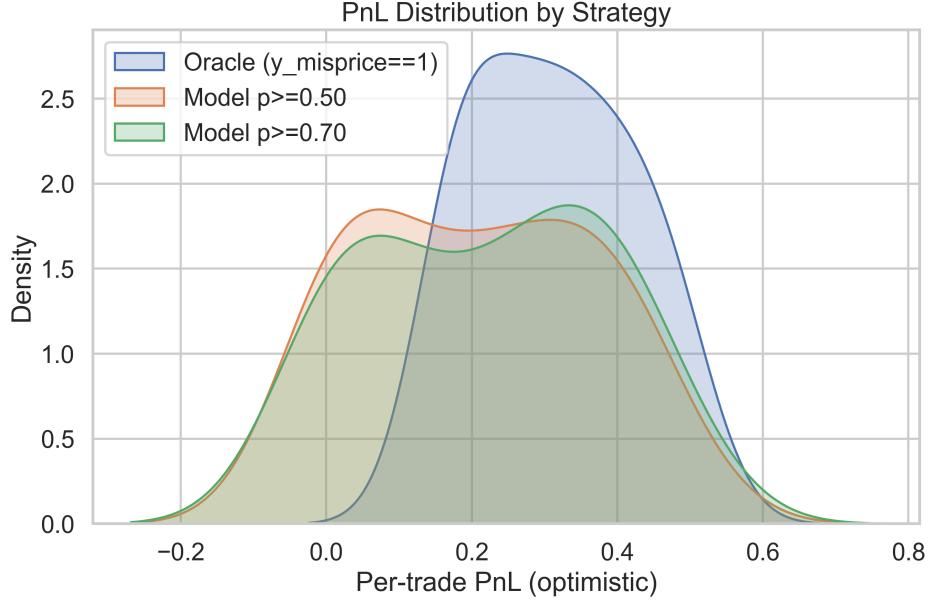


Figure 14: Per-trade PnL distributions. The 0.70 strategy trades less often but with a tighter, more favorable distribution; the 0.50 strategy trades more frequently with a broader spread.

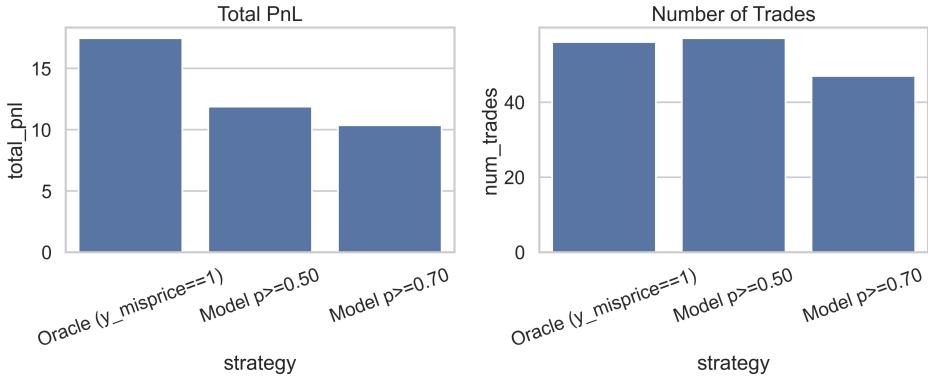


Figure 15: Total PnL and trade counts. The conservative threshold sacrifices volume for precision; the balanced threshold captures more opportunities with more noise.

This analysis does not prove profitability in real markets, but it does show that the model’s scores correlate with larger subsequent price moves in the right direction.

10 Conclusion

Reframing from outcome prediction to mispricing detection produced a model that surfaces short-lived inefficiencies in Polymarket. A tuned XGBoost classifier delivers the best balance of precision and recall under heavy imbalance, with ROC-AUC around 0.94 and F1 around 0.59 at a 0.50 threshold. Interpretation links the signal to intuitive dynamics: volatility spikes, asymmetric order flow, and looming resolution all coincide with higher mispricing risk. Threshold tuning and a simple backtest show how to trade off coverage against confidence.

Limitations remain. The backtest is optimistic and omits execution costs, slippage, and depth. The split is stratified rather than chronological, so live deployment would need walk-forward validation to guard against temporal leakage. Market regimes change, so models require monitoring and retraining, and any real system would need guardrails around liquidity and risk.

Future extensions include rolling validation, richer on-chain or liquidity features, and a live paper-trading loop to observe how calibrated probabilities and thresholds behave in real time. With those additions, this pipeline could evolve from an exploratory research project into a practical tool for monitoring prediction markets.

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.