Predicting Prediction Markets: A Beta-Hidden Markov Modeling Approach

Conrad Oskar Voigt

A thesis presented for the degree of
Bachelor of Science
Advisors: Thomas Vogel, PhD & Lisa Coulter, PhD

Department of Mathematics and Computer Science
Stetson University
April 28, 2025

Abstract

This thesis investigates the informational efficiency of decentralized prediction markets by developing and evaluating probabilistic time series models tailored to the unique structural features of these platforms. Using high-frequency data from Polymarket, the study introduces a novel Groupwise Beta Hidden Markov Model (Beta-HMM) that jointly models interdependent contracts within each market. The model captures latent regime dynamics—such as speculative trading and convergence behavior—while accommodating the bounded and mutually exclusive nature of prediction market prices.

Empirical results demonstrate that the Groupwise Beta-HMM substantially outperforms both a naive benchmark and a standard Beta-HMM, achieving 89.3% classification accuracy and an average per-trade profit of \$0.101. A convergence time metric is introduced to measure belief stabilization, revealing asymmetries in how quickly markets resolve to "Yes" versus "No," with "No" markets converging significantly faster. The paper further explores the practical constraints of implementing such a strategy, including transaction costs, real-time data limitations, and regulatory considerations.

Overall, this work contributes to the literature by proposing a statistically coherent and economically interpretable modeling framework for prediction markets. It highlights the potential of structured probabilistic methods to enhance real-time decision-making and belief aggregation in alternative financial ecosystems.

Contents

Chapter 1

Preliminaries

1.1 Introduction

Prediction markets have emerged as powerful tools for forecasting future events by aggregating dispersed information through financial incentives. By allowing users to buy and sell contracts tied to event outcomes, these markets transform individual beliefs into real-time probability estimates [47, 2]. While prior research has shown that such markets often yield accurate forecasts [4], their informational efficiency—particularly on decentralized platforms with minimal institutional participation—remains an open empirical question.

This paper investigates the informational and economic efficiency of decentralized prediction markets through the lens of time series modeling. We focus on Polymarket, a blockchain-based platform where users speculate on a diverse range of binary and multi-outcome events. Unlike traditional financial markets, Polymarket features minimal market-making infrastructure, heterogeneous trader behavior, and limited liquidity—conditions that challenge classical assumptions of rational pricing and efficient aggregation [48].

To analyze these markets, this paper develop and evaluate a family of Hidden Markov Models (HMMs) adapted for the unique characteristics of prediction market data: bounded price dynamics, regime-switching behavior, and structural heterogeneity across contracts. Building on the Beta-HMM framework [43], the researcher introduce a novel Groupwise Beta-HMM that jointly models related contracts within a market using a masked emission structure, allowing for pooled training across markets with different outcome sets [5, 32].

In addition to predictive classification, a convergence time metric is introduced to assess how quickly markets stabilize around correct outcomes. The analysis highlights both the promise and limitations of statistical arbitrage in prediction markets and suggests structural asymmetries in how "Yes" versus "No" outcomes are priced over time. Overall, the paper contributes to the growing literature on forecasting, belief dynamics, and information aggregation in decentralized financial ecosystems.

1.2 Prediction Markets and their Importance

Prediction markets are speculative platforms where participants trade contracts whose payouts depend on the outcome of future events. Each contract is tied to a binary proposition—such as “Will candidate X win the election?”—and is priced between 0 and 1, reflecting the market’s implied probability of the event occurring. If the event occurs, the contract pays out 1; otherwise, it pays 0. As a result, contract prices can be interpreted as real-time forecasts of collective belief, shaped by the aggregated information and incentives of market participants [47].

Over the past decade, prediction markets have gained considerable traction, particularly with the advent of decentralized, blockchain-based platforms such as Polymarket. These platforms have democratized access by lowering barriers to entry, allowing retail users to participate in forecasting events across politics, finance, science, and culture. As of 2024, several prediction markets routinely surpass $10 million in trading volume, indicating substantial user engagement and increasing relevance in the broader forecasting landscape [38, 4].

What makes prediction markets particularly compelling for empirical study is their structural novelty. Unlike traditional financial markets dominated by institutional investors, prediction markets typically lack professional market makers and are instead populated by retail traders with varying levels of expertise. This composition raises important questions about the efficiency of such markets. Do they aggregate information effectively? Are prices consistent with rational expectations? Or are there persistent inefficiencies that a sophisticated trader or model could exploit [48, 45]?

From a financial perspective, prediction markets are also notable for their near-zero correlation with traditional asset classes. This characteristic has taken on increased significance in today’s volatile economic environment, where equity and fixed-income markets face heightened uncertainty due to inflation, monetary tightening, and geopolitical instability. In such conditions, investors seek alternative assets with uncorrelated payoffs to diversify risk and enhance return potential [18]. The unique information dynamics and idiosyncratic risk structure of prediction markets position them as an attractive domain for both academic research and speculative trading strategies.

In sum, prediction markets offer a rich testbed for studying market efficiency, belief formation, and alternative investment opportunities. Their growth, accessibility, and decoupling from traditional financial markets underscore their importance as both a practical tool for forecasting and a subject of rigorous empirical analysis.

1.3 Data Collection and Preprocessing

The dataset is sourced from Polymarket [38], a blockchain-based prediction market platform. Data was obtained by filtering for resolved bets, selecting only markets with a total trading volume exceeding $10 million. Individual CSV files were downloadable from the platform; however, historical data was only retained for three months at an hourly frequency. Consequently, at each point of download—November 3rd, 2024, and February 17th, 2025—the available data extended back approximately three months from the respective retrieval date. This dual collection approach ensured broader market coverage while mitigating the limitations imposed by the platform’s short data retention policy.

To provide a descriptive overview of the dataset, Table 1.1 summarizes average statistics across 1366 resolved markets. On average, markets contain approximately 487 hourly observations, reflecting the three-month historical window per the data retention policy. The mean and median market-implied probabilities are closely aligned at 0.2341 and 0.2242, respectively, suggesting a mild skew toward lower probability outcomes. The variance and standard deviation—0.0098 and 0.0635—indicate moderate dispersion around the mean, consistent with the inherent volatility of prediction market prices prior to resolution. Notably, while the minimum and maximum prices span nearly the entire probability range [0,1], most contracts cluster around lower probabilities, aligning with the observed imbalance between outcome resolutions: 316 contracts resolved to "Yes" compared to 1050 contracts resolving to "No." The average starting price of 0.2969 and ending price of 0.2323 further reflect the typical decay in price as markets resolve toward "No," supporting the need for a modeling framework that accounts for dynamic belief updating and convergence behavior.

Average Statistic	**Value**
Mean	0.2341
Median	0.2242
Variance	0.0098
Standard Deviation	0.0635
Minimum	0.0005
Maximum	0.9995
Number of Observations	486.99
Starting Value	0.2969
Ending Value	0.2323
Number of Outcome "Yes"	316
Number of Outcome "No"	1050

Table 1.1: Average Summary Statistics Across 1366 Different Markets

The preprocessing of the dataset follows a structured pipeline to ensure the data is clean, consistent, and suitable for time series analysis. The raw data consists of multiple CSV files, each corresponding to a different prediction market. To integrate these sources into a unified dataset, all files are loaded, and their timestamps are standardized by converting the "Date (UTC)" column into a datetime index. This ensures that all market data aligns temporally. During this merging process, columns are renamed with their respective market names to prevent naming conflicts, and duplicate indices are removed to maintain data integrity. The merging operation uses an outer join, ensuring that all data points are preserved while retaining the structure necessary for cross-market comparisons [37].

Once merged, the dataset is resampled to hourly intervals. This step is crucial for enforcing a uniform temporal structure, which is a prerequisite for many time series modeling techniques [34]. Without consistent spacing, models could suffer from irregular gaps that distort temporal dependencies. Missing timestamps are introduced as necessary, and rather than interpolating these values, they are explicitly labeled as "N/A" to prevent artificial bias. This approach is particularly important given the nature of prediction markets, where inactivity at certain time points can be meaningful rather than indicative of data loss [8].

To facilitate downstream analysis, a categorical outcome variable (outcome yes) is introduced to encode whether an event ultimately occurred. This variable is determined based on the final recorded price of each market. If the last observed price exceeds 0.9, the outcome is assigned a value of 1, indicating that the event occurred. Conversely, if the final price is below 0.1, the outcome is assigned 0, indicating that the event did not happen. Cases where the final price falls between these thresholds remain unclassified, as their outcome is ambiguous. This classification is appended as a new row at the top of the dataset, establishing a reference point for supervised learning and predictive modeling applications [43].

Further refinements are applied to eliminate excessive noise in the dataset. Inactive markets often contain long sequences of 0s or 1s at the beginning of their time series, contributing little to predictive modeling. To address this, early observations are truncated while preserving the last 50 data points before the first significant price movement. This prevents the dataset from being dominated by irrelevant periods of inactivity while ensuring that meaningful historical context is retained. Additionally, the timestamp column is replaced with an observation number, allowing for more efficient numerical processing in models that do not require explicit date indexing [5]. To finalize the preprocessing, all empty cells are removed, and remaining values are shifted up to maintain a compact and computationally efficient dataset.

By the end of this process, the dataset is fully structured for analysis, free from duplicate timestamps, standardized to hourly intervals, and labeled for outcome classification. These steps ensure that the data is both clean and meaningful, preserving its integrity while making it suitable for advanced time series modeling techniques.

1.4 Data Features and Model Selection

The unique characteristics of prediction market data inform the choice of modeling framework. The observed price series exhibit several structural properties that must be accounted for in order to build an effective forecasting model.

First, all observations are bounded between 0 and 1, as they represent market-implied probabilities of binary outcomes. Moreover, as the resolution of the event approaches, these prices tend to converge sharply to either 0 or 1, reflecting increasing certainty in the eventual outcome. This behavior is comparable to the pricing of near-expiry binary options, where prices asymptotically approach their payoff values as uncertainty diminishes over time [25].

Second, the data exhibit distinct regime-switching behavior. During the early and mid-phases of trading, market prices often follow a pattern resembling a noisy random walk—driven by changing sentiment, partial information, and speculative activity. Once sufficient information accumulates or a decisive signal emerges, prices rapidly transition to a convergence phase, steadily approaching the final payout value of 0 or 1. These two regimes—*general trading* and *convergence*—are well captured by a Hidden Markov Model (HMM), which explicitly models unobserved states that evolve over time [39]. At a minimum, the model requires two hidden states: one representing the general trading phase and another capturing convergence behavior, which may itself be decomposed into convergence to "Yes" (1) and convergence to "No" (0).

Third, many prediction markets contain groups of related contracts that are logically interdependent. For example, in an election market, separate contracts may exist for each candidate, yet only one outcome can ultimately occur. This introduces structural correlations—often strong negative correlations—between contract prices within the same market. To account for such groupwise dependencies, an extension of the standard HMM is required. Groupwise or coupled HMMs, which allow for coordinated dynamics across multiple related time series, are better suited to capturing the joint evolution of prices in interdependent markets [21, 7].

In light of these data features, a Beta-HMM with multiple hidden states is selected to model the bounded and regime-switching nature of the time series [43]. Extensions of this model to groupwise settings are discussed in later sections to better reflect correlated market structures.

1.5 Model Task and Benchmarking

The primary task of the model is to predict the eventual outcome of a binary prediction market before the market is resolved. Each market resolves to either ”Yes” (1) or ”No” (0), and the profit is calculated as the difference between the final payoff (which is always \$1 for a correct position) and the price at which the position was entered. This framing allows the model to be evaluated not only by classification accuracy but also by its economic performance in a trading context.

The trading strategy employed is a long-only approach. This means the model only takes long positions—either buying ”Yes” or buying ”No”—and does not engage in short-selling or borrowing. This design choice avoids borrowing costs and reflects the structure of most prediction markets, where reciprocal contracts exist: if a ”Yes” contract is available for purchase, a ”No” contract is also typically available, both priced to sum to approximately \$1. As such, taking a long-only position on either side suffices to represent both sides of the market without requiring additional complexity.

To evaluate the performance of the model, it is compared against a simple naive benchmark. The benchmark strategy operates without any predictive modeling. Instead, it places a bet exactly eight hours before market closure by taking a long position on whichever side—”Yes” or ”No”—has the higher observed market price at that time. This approach serves as a baseline to assess whether the Beta-HMM provides meaningful improvements in both predictive accuracy and trading profitability over a non-model-driven heuristic.

Chapter 2

Convergence Time Analysis in Prediction Markets

2.1 Motivation and Definition

Prediction markets display distinct temporal patterns in the evolution of prices. In the early phases of a market, contract prices fluctuate significantly, driven by speculation, information asymmetries, and noise. As the event approaches resolution and uncertainty decreases, these prices tend to stabilize around the final realized outcome. This stabilization behavior—referred to as *convergence*—is a key empirical feature of functioning prediction markets, as it reflects the market's ability to efficiently aggregate information over time [47, 4].

Figure 2.1 presents a representative example from Polymarket: a multi-outcome market forecasting the next UK Conservative Party leader. Initially, contract prices are dispersed and volatile. Over time, however, one contract (orange) begins to rise steadily, while others decline, indicating increasing market consensus. By the end of the series, the dominant contract approaches a price of one, while others fall toward zero—highlighting a clear instance of convergence.

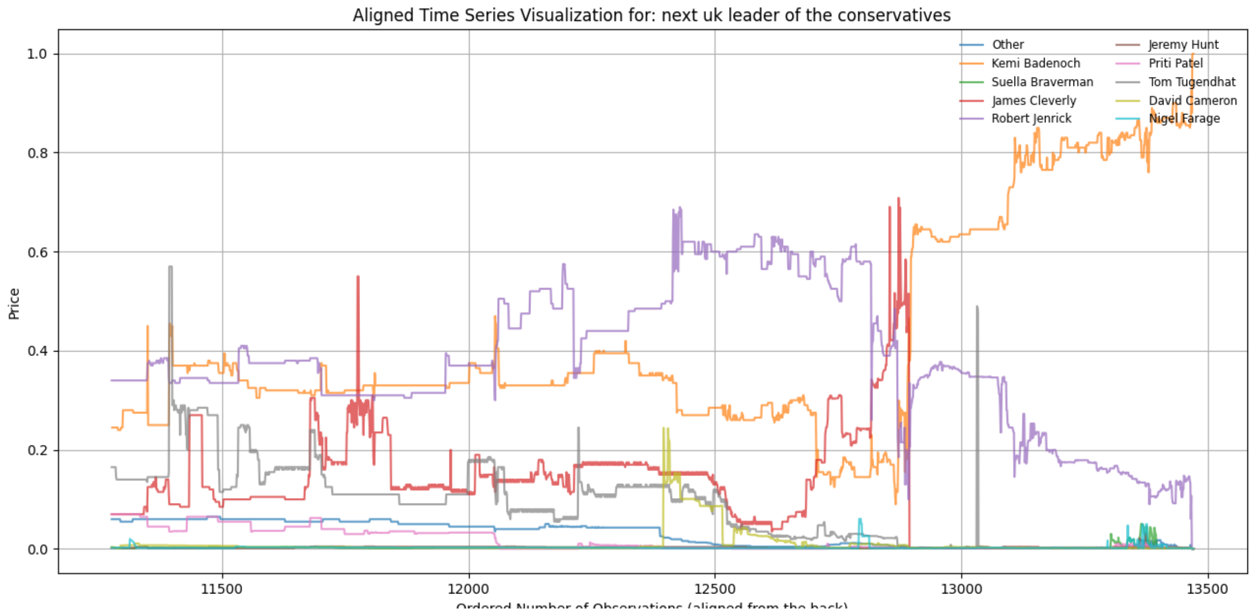


Figure 2.1: Example of price convergence in a multi-outcome prediction market (Polymarket: next UK Conservative leader).

To quantify this phenomenon, the **forward convergence time** statistic τ_ε is defined. Given a market-implied probability sequence $\{p_t\}_{t=1}^{T}$ and a realized binary outcome $y \in \{0,1\}$, the convergence time is defined as:

$$\tau_\varepsilon = \min\{t \in \{1,\dots,T\} : |p_s - y| < \varepsilon \text{ for all } s \geq t\}$$

Here, $\varepsilon \in (0,1)$ specifies the width of the convergence band, such as 0.10. The statistic τ_ε captures the earliest point from which the market remains consistently close to the correct outcome.

Convergence time is a useful metric in this context for three primary reasons. First, it enables rigorous cross-market comparisons of informational efficiency by measuring how quickly beliefs stabilize. Second, the existence of sharp transitions from noisy trading to stable convergence motivates the adoption of regime-switching models—such as Hidden Markov Models—that can capture latent state dynamics [8]. Third, convergence timing has practical trading implications: early convergence may enable profitable prediction-based positions before resolution.

2.2 Empirical Analysis of Market Convergence

An empirical investigation of convergence behavior in decentralized prediction markets is conducted by evaluating the forward convergence time statistic τ_ε across a panel of resolved binary event markets. The convergence metric τ_ε is computed for three tolerance levels $\varepsilon \in \{0.05, 0.10, 0.15\}$, and the distributional behavior is examined via kernel density estimation (KDE) to visualize the distribution of τ_ε across resolved markets.

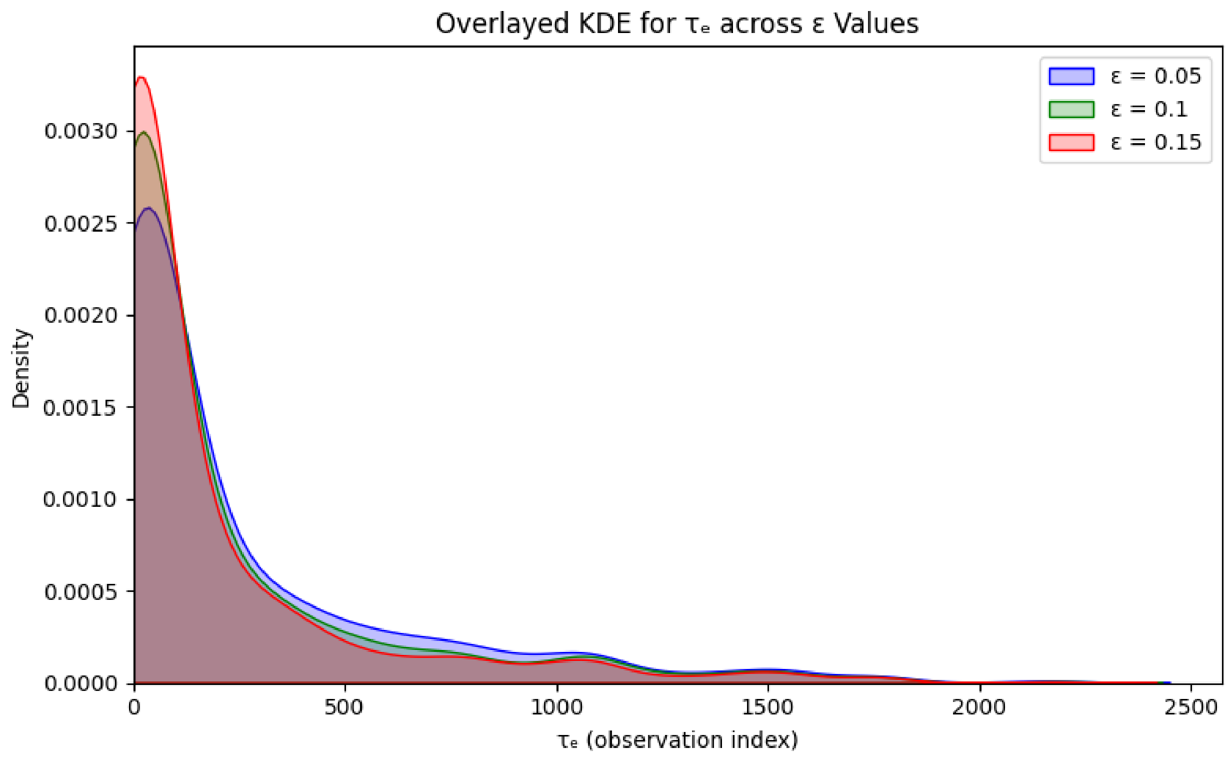


Figure 2.2: Overlayed KDE plots for different Taus

2.2.1 Distribution of Convergence Times

Figure 2.2 presents the KDEs of τ_ε across all markets for three tolerance levels $\varepsilon \in \{0.05, 0.10, 0.15\}$. Across all thresholds, the distribution exhibits a long right tail and a sharp mode near zero, suggesting that most markets converge very early, but a non-negligible subset exhibit delayed or unstable convergence.

As expected, the average convergence time declines as ε increases: mean values of 255.7, 211.4, and 186.9 for $\varepsilon = 0.05, 0.10, 0.15$, respectively. This inverse relationship reflects the definition of τ_ε: wider tolerance bands allow earlier entry into the stability region. The interquartile range also shrinks with increasing ε, indicating tighter dispersion and faster convergence.

ε	Mean	Median	Std Dev	Var	25%	75%
0.05	255.66	87.00	387.17	149,901.65	1.00	329.50
0.10	211.42	42.00	361.25	130,498.44	0.00	242.50
0.15	186.85	21.00	346.17	119,834.09	0.00	172.50

Table 2.1: Descriptive Statistics for τ_ε Across Tolerance Levels

2.2.2 Stratification by Outcome

To assess whether convergence dynamics differ by resolution outcome, this paper stratifies $\tau_{0.10}$ values into markets that resolved to "Yes" ($y = 1$) and those that resolved to "No" ($y = 0$). Figure 2.3 overlays the KDEs of the two subgroups.

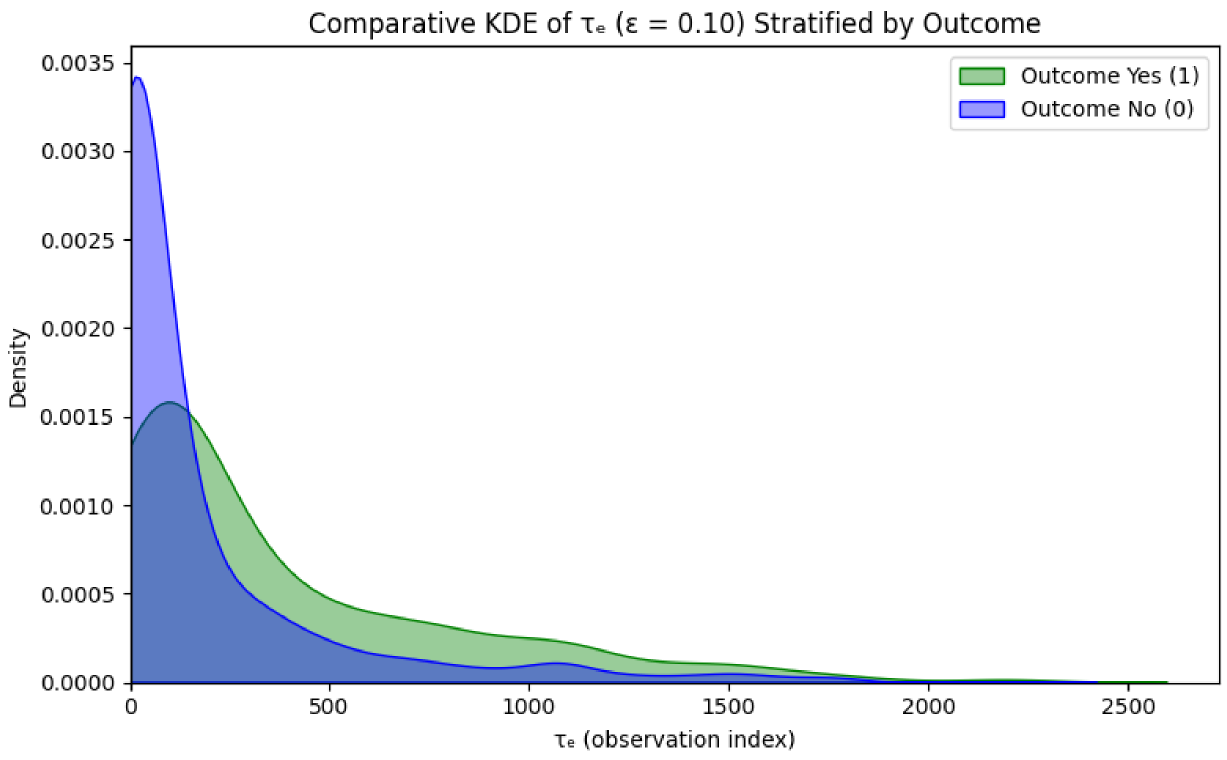


Figure 2.3: KDE of $\tau_{0.10}$ Stratified by Outcome

The distributions diverge markedly. Markets that resolved to "Yes" exhibit significantly longer convergence times on average (mean: 371.8 vs. 171.3), with a broader spread (std dev: 438.6 vs. 327.1). Moreover, the median $\tau_{0.10}$ for "No" markets is just 7, compared to 167.5 for "Yes," indicating that "No" markets converge almost immediately in many cases. This asymmetry may reflect inherent biases or risk-aversion in market participation, where the burden of proof—or information revelation—is higher for "Yes" outcomes.

Outcome	Mean	Median	Std Dev	Var	25%	75%
Yes (1)	371.77	167.50	438.57	192,342.68	51.25	557.00
No (0)	171.31	7.00	327.09	106,985.06	0.00	167.00

Table 2.2: Descriptive Statistics for $\tau_{0.10}$ by Outcome

2.2.3 Tolerance Sensitivity of Convergence Time

Figure 2.4 displays the mean convergence time τ_ε as a function of the tolerance level ε, separately for the aggregate sample and for markets resolving to "Yes" and "No." Across all three groups, the empirical relationship is strictly decreasing: as the tolerance band widens from $\varepsilon = 0.01$ to $\varepsilon = 0.20$, the mean τ_ε falls monotonically. This behavior is a direct consequence of the definition in Equation (**??**), since larger values of ε impose a less stringent requirement for "locking in" the series within the neighborhood of the realized outcome.

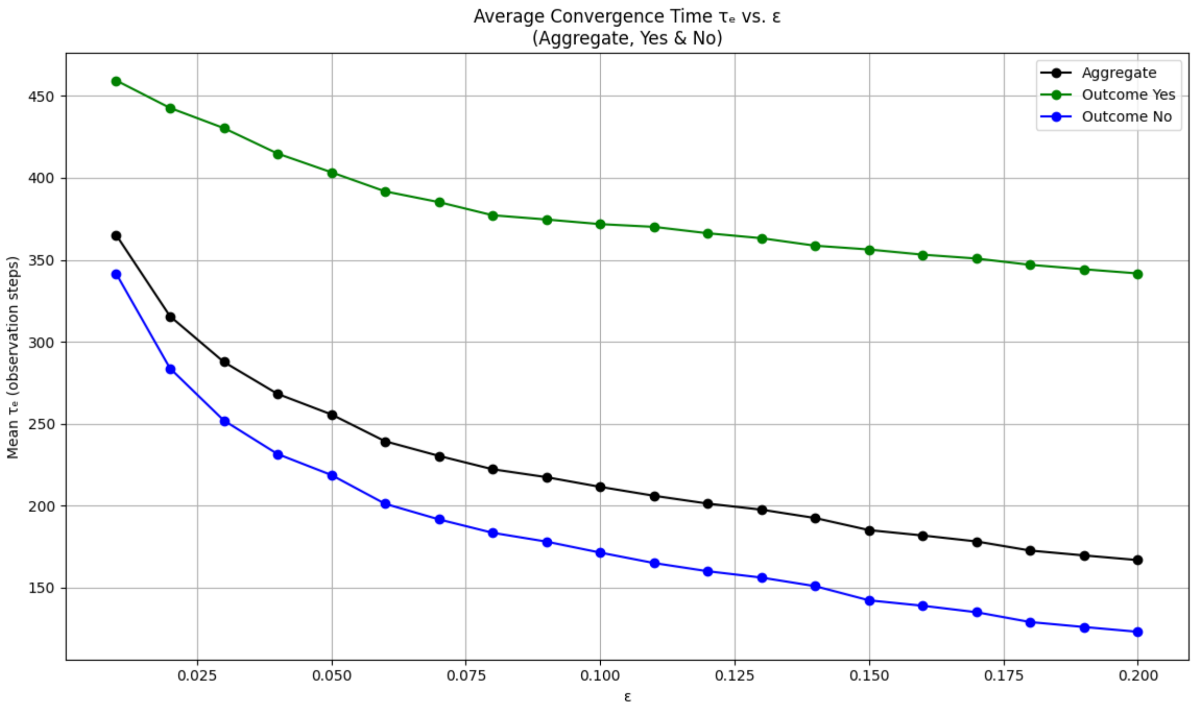


Figure 2.4: Tau vs Epsilon

In the aggregate series (black curve), the decline in mean τ_ε is steepest at small tolerances, dropping from approximately 365 observation-steps at $\varepsilon = 0.01$ to around 230 steps by $\varepsilon = 0.05$, before gradually flattening to roughly 168 steps at $\varepsilon = 0.20$. This convex shape indicates diminishing returns to increasing ε: initial relaxations of the tolerance yield large gains in convergence speed, whereas further widening produces smaller marginal effects.

The stratified curves reveal a pronounced asymmetry between outcome regimes. Markets resolving to "No" (blue curve) exhibit systematically faster convergence than those resolving to "Yes" (green curve) at every tolerance level. For instance, at $\varepsilon = 0.05$, the mean $\tau_{0.05}$ for "No" markets is approximately 218 steps, compared to about 402 steps for "Yes" markets. Even at the largest tolerance of $\varepsilon = 0.20$, the "No" curve attains an average $\tau_{0.20} \approx 123$, whereas the "Yes" market still requires nearly 342 steps on average. This persistent gap suggests that baseline (negative) outcomes are assimilated more quickly by the market, potentially reflecting participant risk-aversion or the relative ease of aggregating disconfirming signals.

Overall, the tolerance-sensitivity analysis underscores two key insights. First, the choice of ε critically shapes the measured convergence speed, with small tolerances producing markedly slower τ_ε estimates. Second, there exists a robust structural asymmetry in belief updating: markets coalesce around "No" outcomes substantially faster than around "Yes," even when evaluated under identical tolerance criteria. These findings highlight both methodological considerations for selecting ε in empirical studies of informational efficiency and substantive implications for the dynamics of market belief formation across different event polarities.

2.3 Relevance to the Modeling Task

2.4 Relevance to the Modeling Task

This chapter investigated how quickly prediction markets stabilize around correct outcomes using a forward convergence time statistic, τ_ε. Empirical analysis across a large panel of resolved binary markets revealed several robust patterns. Most notably, the distribution of convergence times is heavily right-skewed: while many markets converge early, a significant minority remain volatile until very late. Moreover, convergence occurs faster and more decisively in markets resolving to "No" than in those resolving to "Yes," suggesting asymmetries in how evidence is incorporated depending on the polarity of the outcome [4, 47].

These results are more than descriptive. The observed patterns of sudden stabilization, especially when preceded by extended speculative periods, strongly suggest the presence of underlying latent regimes. Such regimes are not directly observable from price data alone but manifest as qualitative shifts in price behavior—from noisy fluctuations to sustained directional movement. This motivates the use of probabilistic models with unobserved state variables that can infer and characterize these hidden dynamics [8, 39].

In the next chapter, the Beta Hidden Markov Model (Beta-HMM) is introduced, which formalizes this intuition. By assigning a latent state to each time point—interpretable as either speculative or converged—the model allows us to detect convergence endogenously and exploit these signals for predictive purposes [43, 5].

Chapter 3

Fundamental Concepts Underpinning the Hidden MArkov Model

The Beta Hidden Markov Model (Beta-HMM) extends the classical Hidden Markov Model (HMM) framework by incorporating Beta-distributed emissions. To fully appreciate its structure, it is crucial to first understand the foundational probabilistic models it builds upon: Markov processes and Markov chains. These serve as the backbone for modeling sequential dependencies in time-series data.

3.1 Markov Processes and Their Properties

A Markov process is a stochastic system where the future state depends only on the present state, independent of the past. More formally, given a sequence of states $S_1, S_2, \ldots, S_T$, the Markov property holds if:

$$P(S_t|S_{t-1}, S_{t-2}, ..., S_1) = P(S_t|S_{t-1}). \tag{3.1}$$

This assumption, often referred to as memorylessness, simplifies analysis while still capturing essential sequential dependencies [40].

A Markov chain is a discrete-time Markov process where both time and state spaces are discrete. Let S_t be a sequence of random variables taking values in a finite state space $\{0, 1, \ldots, J-1\}$. The system transitions between states according to a transition probability matrix (TPM) A, where:

$$A_{ij} = P(S_{t+1} = j|S_t = i). \tag{3.2}$$

If the transition probabilities remain constant over time, the chain is said to be homogeneous, and the matrix A satisfies the stochastic property [35, 28]:

$$\sum_{j=0}^{J-1} A_{ij} = 1, \quad \forall i \in \{0, \ldots, J-1\}. \tag{3.3}$$

3.2 Short-Term and Long-Term Dynamics of Markov Chains

The short-term evolution of a Markov chain is characterized by its one-step transition matrix A. However, long-term behavior is determined by the k-step transition probabilities:

$$P(S_{t+k} = j | S_t = i) = (A^k)_{ij}. \tag{3.4}$$

This relation allows us to study the long-term equilibrium distribution of the system [14].

A state j is said to be accessible from state i (denoted $i \to j$) if there exists some $k \geq 1$ such that $(A^k)_{ij} > 0$. If both $i \to j$ and $j \to i$ hold, the states are said to communicate (denoted $i \leftrightarrow j$). The Markov chain is called irreducible if all states communicate, ensuring no subset of states is isolated [23].

3.3 Convergence to the Stationary Distribution and Mixing Time

The long-term behavior of a Markov chain is governed by its stationary distribution π, which satisfies the fixed-point equation:

$$\pi A = \pi, \quad \text{subject to } \sum_{i=0}^{J-1} \pi_i = 1. \tag{3.5}$$

This distribution represents the limiting probabilities of being in each state as $t \to \infty$. When the Markov chain is irreducible and aperiodic, a unique stationary distribution exists and is reached regardless of the initial state distribution [30].

A powerful way to understand convergence is through the spectral decomposition of the transition matrix A. Suppose A has eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_J$ with corresponding eigenvectors $v_1, v_2, \ldots, v_J$. Then:

$$A^k = QD^kQ^{-1}, \tag{3.6}$$

where Q is the matrix of eigenvectors, and D^k is a diagonal matrix with entries λ_i^k. Since stochastic matrices always have a largest eigenvalue $\lambda_1 = 1$, and all other eigenvalues satisfy $|\lambda_i| < 1$ for $i \geq 2$, repeated matrix multiplication leads to:

$$\lim_{k\to\infty} A^k = v_1 \pi^T, \tag{3.7}$$

where v_1 is the right eigenvector corresponding to $\lambda_1 = 1$, and π is its associated left eigenvector, normalized to sum to one [19]. This convergence guarantees that, in the long run, the system settles into a steady-state behavior characterized by π.

The speed at which this equilibrium is reached is captured by the *mixing time*, defined as:

$$t_{\text{mix}}(\epsilon) = \min \left\{ t : \max_i \sum_j |P(S_t = j | S_0 = i) - \pi_j| < \epsilon \right\}. \tag{3.8}$$

This quantity measures how quickly the distribution over states becomes indistinguishable from the stationary distribution within a margin ϵ. The rate of convergence is largely determined by the second-largest eigenvalue modulus (SLEM), given by:

$$\rho = \max\{|\lambda_2|, |\lambda_3|, \ldots, |\lambda_J|\}. \tag{3.9}$$

A smaller ρ implies faster decay of transient dynamics and quicker convergence to the stationary distribution. This is particularly relevant in applications such as financial modeling, where rapid inference and adaptability to new information are essential for real-time decision-making [33].

3.4 Relevance to the Framework

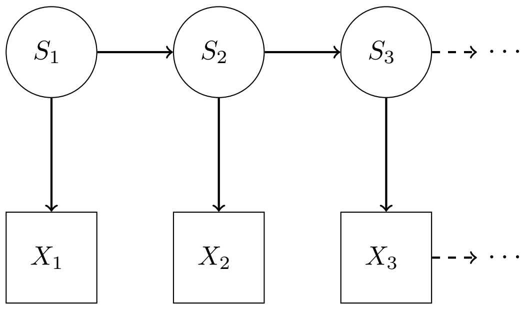
The theoretical properties of Markov chains—such as accessibility, stationary distributions, and convergence behavior—form the backbone of more complex probabilistic models used in sequential data analysis [35, 15]. In particular, the understanding of how hidden states evolve over time, stabilize, or shift between regimes is essential when these states are not directly observed but instead inferred from data.

This long-run structure becomes especially important when the Markov chain is embedded within a larger modeling framework in which observations depend probabilistically on latent state dynamics [8]. In such settings, the Markov chain governs the temporal evolution of unobserved regimes, while the observed data provide indirect evidence about the system's current state.

The insights gained from studying short- and long-term Markov chain dynamics—such as mixing time, spectral properties, and steady-state behavior—directly inform the design, estimation, and interpretation of such models [30].

Chapter 4

The Beta Hidden Markov Model



$$\begin{tikzcd}
S_1 \arrow[r] \arrow[d] & S_2 \arrow[r] \arrow[d] & S_3 \arrow[r, dashed] \arrow[d] & \cdots \\
X_1 & X_2 & X_3 \arrow[r, dashed] & \cdots
\end{tikzcd}$$

Figure 4.1: Graphical representation of a Hidden Markov Model (HMM), where hidden states S_t generate observed emissions X_t.

A Hidden Markov Model (HMM) is a probabilistic model that represents a system evolving over time through an unobservable sequence of hidden states, which influence an observable sequence of emissions [39, 8]. The key characteristic of an HMM is that while the system undergoes Markovian transitions between hidden states, the true states themselves remain unknown and must be inferred from observations.

Traditionally, HMMs assume Gaussian or Poisson-distributed emissions, making them well-suited for unbounded continuous or count-based data, respectively [5, 13]. However, in scenarios where observed values are constrained within a fixed range, such as probabilities in the interval $[0, 1]$, the Gaussian assumption becomes problematic due to its support over the entire real line. In such cases, the Beta distribution provides a more suitable alternative, leading to the formulation of the Beta-HMM.

The Beta-HMM extends the classical HMM framework by incorporating Beta-distributed emissions, making it particularly well-suited for modeling probabilistic outcomes such as market beliefs, proportions, and probabilities of binary events [43].

4.1 Model Components and Initialization

The Beta-HMM consists of two core sequences: a hidden state sequence S_t and an observable sequence X_t, where each X_t is conditionally dependent on the hidden state S_t at time t. The model follows the Markov property, ensuring that future states depend only on the current state and not on past states [34].

A standard HMM is formally defined by three primary components: the state transition matrix A, which encodes the probabilities of transitioning between hidden states; the initial state distribution δ, which specifies the probability of starting in each state; and the emission probability distribution $f(X_t|S_t)$, which governs how observable data are generated conditional on the current hidden state.

4.1.1 State Transition Matrix

The state transition matrix A governs the probabilities of transitioning between the three hidden states: **general trading** (S_0), **yes convergence** (S_1), and **no convergence** (S_2). The transition probabilities are defined as:

$$A_{ij} = P(S_t = j \mid S_{t-1} = i), \tag{4.1}$$

where A is a stochastic matrix, meaning each row sums to 1:

$$\sum_{j=0}^{2} A_{ij} = 1, \quad \forall i \in \{0, 1, 2\}. \tag{4.2}$$

Since the transition probabilities remain constant over time, the model assumes a time-homogeneous Markov process, ensuring consistency in sequence modeling [39].

In matrix form, the state transition matrix is:

$$A = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}, \tag{4.3}$$

where:

- A_{00} represents the probability of remaining in the **general trading** state,
- A_{01} and A_{02} denote the probabilities of transitioning from **general trading** to **yes convergence** and **no convergence**, respectively,
- A_{10} and A_{20} capture reversions from **yes convergence** and **no convergence** back to **general trading**,
- A_{11} and A_{22} are the self-transition probabilities for the absorbing states **yes convergence** and **no convergence**,

- A_{12} and A_{21} represent cross-transitions between **yes convergence** and **no convergence**, which are expected to be minimal in a well-behaved market.

Since A is a stochastic matrix, it satisfies:

$$\sum_{j=0}^{2} A_{ij} = 1, \quad \forall i \in \{0, 1, 2\}. \tag{4.4}$$

The transition matrix defines the probabilistic evolution of market phases over time. It encodes dependencies between past and future states, playing a critical role in modeling the dynamics of prediction market movements. The stationary distribution π, obtained as the left eigenvector satisfying $\pi A = \pi$, provides insights into the long-run equilibrium distribution of market states, which is useful for forecasting market trends and price behavior [39].

4.1.2 Initial State Distribution

The initial probability distribution δ describes the likelihood of starting in each state:

$$\delta_i = P(S_1 = i), \tag{4.5}$$

with the constraint:

$$\sum_i \delta_i = 1. \tag{4.6}$$

4.1.3 Emission Distributions

A classical HMM assumes Gaussian emissions:

$$X_t | S_t = i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{4.7}$$

[5, 13]. However, Gaussian emissions are unsuitable when modeling data constrained to $[0, 1]$, such as implied probabilities, like in our case. Since Gaussian distributions extend over the entire real line, they can produce infeasible values, leading to biased or invalid inferences.

To overcome this limitation, we introduce the Beta distribution as the emission distribution:

$$X_t | S_t = i \sim \text{Beta}(\alpha_i, \beta_i) \tag{4.8}$$

[26]. The Beta probability density function (PDF) is given by:

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \quad 0 < x < 1, \tag{4.9}$$

where $B(\alpha, \beta)$ is the Beta function, defined as:

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt. \tag{4.10}$$

The Beta function is closely related to the Gamma function, which generalizes the factorial function for noninteger values. Specifically, the Beta function can be expressed in terms of Gamma functions as follows[1]:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}, \tag{4.11}$$

where the Gamma function is defined as:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt. \tag{4.12}$$

This relationship is fundamental in many probabilistic and statistical applications, as it allows the Beta function to be computed using well-known properties of the Gamma function. The connection to the Gamma function also simplifies parameter estimation in Bayesian inference, as conjugate priors involving the Beta distribution can be expressed in terms of Gamma-distributed hyperparameters [20].

By leveraging this property, we can efficiently compute the Beta distribution parameters in a Hidden Markov Model while ensuring that emissions remain properly bounded within the interval $[0, 1]$. The flexibility of the Beta distribution, combined with its connection to the Gamma function, makes it a natural choice for modeling probability-like data in a structured sequential framework.

4.2 Advantages of the Beta Distribution

The Beta distribution offers several advantages over the Gaussian model. First, it provides bounded support, ensuring that emissions remain within the $[0, 1]$ range—unlike Gaussian models, which can produce infeasible values outside this interval.

Second, the Beta distribution is highly flexible, capable of modeling a wide variety of shapes, including uniform, unimodal, and bimodal distributions, depending on its parameters α and β.

Third, the distribution is interpretable, as its parameters can be expressed directly in terms of the mean and variance:

$$\mu = \frac{\alpha}{\alpha + \beta}, \quad \sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}. \tag{4.13}$$

These properties make the Beta distribution particularly useful for probabilistic modeling in finance, where observed market beliefs and probabilities naturally fall within the $[0, 1]$ interval [43].

Empirical validation supports the use of Beta-distributed emissions. Figure 4.2 presents a histogram of observed market probabilities overlaid with a fitted Beta distribution. Compared to a Gaussian fit, the Beta distribution more accurately captures the skewness and bounded nature of financial probability distributions.

By replacing Gaussian emissions with Beta-distributed emissions, the Beta-HMM extends the traditional HMM framework to handle bounded data more effectively. The choice of Beta-distributed emissions is mathematically justified and empirically validated, making the Beta-HMM particularly relevant for financial and probabilistic forecasting.

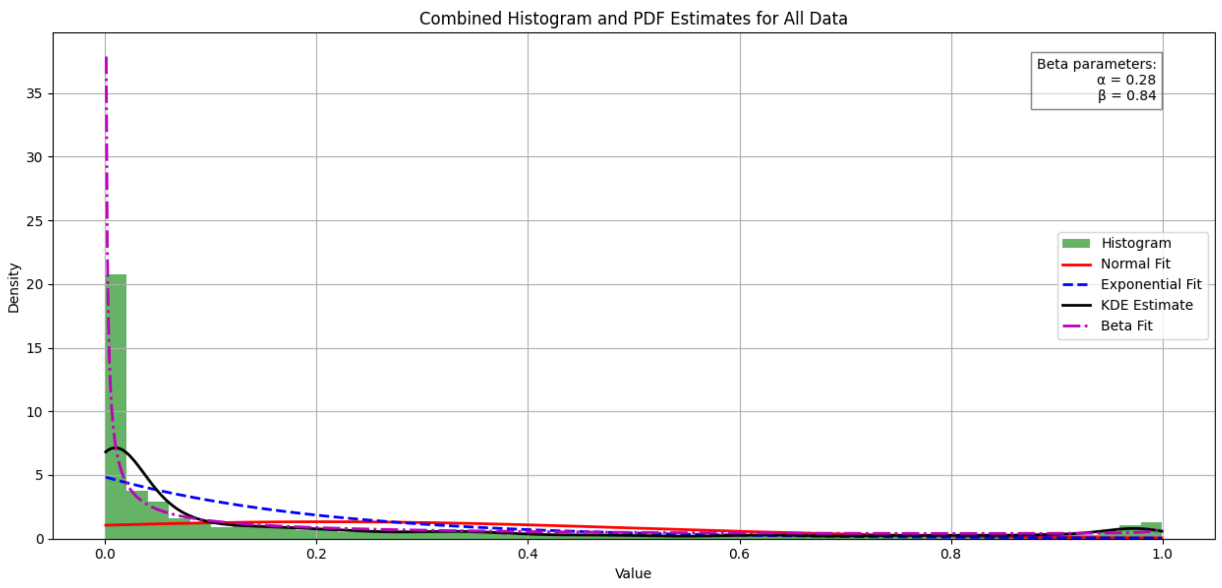


Figure 4.2: Histogram and Fitted Beta Distribution

4.3 Parameter Initialization

The model parameters are initialized as follows. The transition probabilities A_{ij} are randomly assigned and then normalized so that each row sums to one. Similarly, the initial state probabilities δ_i are randomly initialized and normalized to ensure that the total probability sums to one. The Beta distribution parameters α and β are either chosen based on prior domain knowledge or estimated from data using the method of moments [9].

4.4 Expectation-Maximization Algorithm (EM) for Parameter Estimation

To estimate the parameters $\lambda = (\delta, A, \alpha, \beta)$, the expectation-maximization (EM) algorithm is employed [12, 5]. The EM algorithm is an iterative optimization procedure that computes maximum likelihood estimates for models involving latent variables or incomplete data. It consists of two alternating steps. In the expectation step (E-step), the expected values of the latent variables are computed based on the current estimates of the parameters. In the subsequent maximization step (M-step), the parameter values are updated to maximize the expected complete-data log-likelihood obtained in the E-step.

4.4.1 Expectation Step (E-Step)

In this step, the expected sufficient statistics are computed based on the current parameter estimates.

The forward probability $\alpha_t(i)$ is defined as the probability of observing the sequence up to time t, given that the process is in state i:

$$\alpha_t(i) = P(X_1, ..., X_t, S_t = i|\lambda) \tag{4.14}$$

The forward probabilities are computed recursively using the forward algorithm. The initialization step is given by:

$$\alpha_1(i) = \delta_i f(X_1|S_1 = i) \tag{4.15}$$

The recursion step is defined as:

$$\alpha_t(j) = \sum_i \alpha_{t-1}(i) A_{ij} f(X_t|S_t = j) \tag{4.16}$$

The backward probability $\beta_t(i)$ represents the probability of observing the remaining sequence given that the process is in state i at time t. The initialization step is:

$$\beta_T(i) = 1 \tag{4.17}$$

The recursion step is:

$$\beta_t(i) = \sum_j A_{ij} f(X_{t+1}|S_{t+1} = j)\beta_{t+1}(j) \tag{4.18}$$

Using these forward and backward probabilities, the posterior probability is computed as:

$$\gamma_t(i) = P(S_t = i|X_1, ..., X_T) = \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)} \tag{4.19}$$

The joint probability of transitioning from state i to j is given by:

$$\xi_t(i,j) = \frac{\alpha_t(i) A_{ij} f(X_{t+1}|S_{t+1} = j)\beta_{t+1}(j)}{\sum_{i,j} \alpha_t(i) A_{ij} f(X_{t+1}|S_{t+1} = j)\beta_{t+1}(j)} \tag{4.20}$$

4.4.2 Maximization Step (M-Step)

In this step, the model parameters are updated using the expectations computed in the E-step.

The transition probability matrix A is updated as:

$$A_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{4.21}$$

The initial probability distribution is updated as:

$$\delta_i = \gamma_1(i) \tag{4.22}$$

The method of moments is used to estimate the parameters α_i and β_i of the Beta distribution [9, 26].

The mean is estimated as:

$$\mu_i = \frac{\sum_t X_t \gamma_t(i)}{\sum_t \gamma_t(i)} \tag{4.23}$$

The variance is estimated as:

$$\sigma_i^2 = \frac{\sum_t \gamma_t(i)(X_t - \mu_i)^2}{\sum_t \gamma_t(i)} \tag{4.24}$$

The Beta distribution parameters are then derived from the estimated mean and variance:

$$\alpha_i = \mu_i \left(\frac{\mu_i(1-\mu_i)}{\sigma_i^2} - 1 \right) \tag{4.25}$$

$$\beta_i = (1-\mu_i) \left(\frac{\mu_i(1-\mu_i)}{\sigma_i^2} - 1 \right) \tag{4.26}$$

4.5 Inference and Prediction using the Viterbi Algorithm

Once the Beta-HMM is trained, the next step is to infer the most likely sequence of hidden states given a new sequence of observations $X_1, X_2, \ldots, X_T$. The goal is to determine the optimal state sequence $S_1^*, S_2^*, \ldots, S_T^*$ that maximizes the posterior probability of the hidden state path, formulated as

$$S_T^* = \arg\max_i P(S_T = i | X_1, ..., X_T). \tag{4.27}$$

Since directly computing all possible state sequences is computationally infeasible due to the exponential complexity $O(N^T)$, an efficient solution is required. The Viterbi algorithm provides an optimal approach based on dynamic programming that determines the most probable hidden state sequence in an HMM while maintaining polynomial complexity in time and space [39, 17, 13, 5].

The derivation of the Viterbi algorithm relies on the principle of optimal substructure, which states that if the most probable sequence leading to state S_t passes through state S_{t-1}, then the sub-path up to S_{t-1} must also be the most probable sub-path up to that point. This property allows for an efficient recursive formulation [34]. The algorithm consists of four steps: initialization, recursion, termination, and backtracking.

At the initial time step $t = 1$, the probability of starting in each hidden state given the first observation is computed as

$$\delta_1(i) = \pi_i f(X_1 | S_1 = i), \quad \forall i. \tag{4.28}$$

Additionally, a pointer matrix is initialized to keep track of the most probable state transitions, defined as

$$\psi_1(i) = 0, \quad \forall i. \tag{4.29}$$

For each subsequent time step $t = 2, ..., T$, the probability of reaching state j through the most probable prior state is updated as

$$\delta_t(j) = \max_i \left[\delta_{t-1}(i) A_{ij} \right] f(X_t | S_t = j), \quad \forall j. \tag{4.30}$$

Simultaneously, the most likely state transition is recorded as

$$\psi_t(j) = \arg\max_i \left[\delta_{t-1}(i) A_{ij}\right], \quad \forall j. \tag{4.31}$$

At the final time step T, the most probable hidden state is determined by

$$S_T^* = \arg\max_i \delta_T(i). \tag{4.32}$$

Once the final state has been identified, the most probable sequence of hidden states is reconstructed by tracing backward through the stored transitions. The backtracking step is given by

$$S_t^* = \psi_{t+1}(S_{t+1}^*), \quad t = T-1, T-2, ..., 1. \tag{4.33}$$

The computational complexity of the Viterbi algorithm is $O(N^2T)$, where N is the number of states and T is the length of the observation sequence. This is significantly more efficient than a brute-force approach that evaluates all possible state sequences, which would require exponential time $O(N^T)$. The main advantage of the dynamic programming approach is that suboptimal paths are pruned early, thereby reducing unnecessary computations. Moreover, since the algorithm only tracks the highest probability path, it prevents overfitting to local fluctuations in the observation sequence [13, 8].

4.6 Extracting Predictions

Once the most probable sequence of hidden states $S_1^*, S_2^*, ..., S_T^*$ has been obtained, the model extracts a final prediction based on the inferred market state. Specifically, the final predicted outcome is determined by identifying the first time step at which the final hidden state S_T^* is first emitted. If $S_T^* = 1$, the model predicts convergence to "Yes"; if $S_T^* = 2$, convergence to "No". This approach ensures that outcome predictions are not based on short-term noise but rather on stable regime identification.

The corresponding market price at the first observation of the final hidden state—denoted P_{entry}—is recorded for profit and loss (P/L) analysis. The realized P/L is computed using the final market resolution price P_{final}, according to the formula:

$$\text{P/L} = P_{\text{final}} - P_{\text{entry}}. \tag{4.34}$$

This procedure provides a systematic and interpretable method for linking hidden state estimates to real-world economic performance metrics.

4.7 Implementation in Python

The Beta-HMM was implemented entirely from scratch in Python, as no existing library provided built-in support for Beta-distributed emissions in the context of Hidden Markov Models. The implementation uses core scientific computing libraries, including `NumPy` for efficient numerical operations and array manipulation, and `Pandas` for data preprocessing and structured dataset handling [36, 37].

The Beta probability density function used in the emission step was sourced from the `scipy.stats.beta` module, which provides a reliable and numerically stable implementation of the Beta distribution [42]. Additionally, `scikit-learn` was used for train-test splitting to facilitate robust evaluation across multiple runs [41].

All code used to construct, train, and evaluate the Beta-HMM is provided in the Appendix.

Chapter 5

The Groupwise Beta Hidden Markov Model

This chapter introduces a groupwise modeling framework that leverages natural clusters of prediction markets whose dynamics co-move. Pearson correlation coefficients are computed on the first-differenced price series for every pair of markets; high correlations signal homogeneous behavior and define the grouping structure. Within each cluster, a linear approximation is then fitted that includes both a group-specific slope and a cross-term interaction, capturing how aggregate trends and idiosyncratic deviations jointly drive convergence. This cross-term enables the model to account for intergroup dependencies and improves fit relative to models that treat markets in isolation [5, 43].

5.1 Data Adjustments and Empirical Motivation for Groupwise Modeling

To reflect structural differences in contract types, the dataset is divided into single-outcome markets (binary resolution) and multi-outcome bets (more than two possible final states). Single-outcome markets are compatible with the standard Hidden Markov framework, whereas multi-outcome markets benefit from the groupwise extension, which accommodates multiple absorbing states. Comparing predictive performance across these two regimes underscores the necessity of tailored methods for different market architectures.

To justify the transition from independent univariate models to a joint groupwise modeling approach, this paper conducts a comprehensive correlation and multicollinearity analysis of contracts within the same prediction market. For each market, all pairwise Pearson correlation coefficients between contracts are computed and the minimum, maximum, mean, and median values are summarized. Multicollinearity is evaluated using the mean Variance Inflation Factor (VIF), which quantifies how much the variance of an estimated regression coefficient increases due to collinearity among predictors. Specifically, for a predictor variable x_j, the VIF is defined as:

$$\mathrm{VIF}_j = \frac{1}{1 - R_j^2}, \tag{5.1}$$

where R_j^2 is the coefficient of determination from regressing x_j on all other predictors. A VIF above

10 is commonly considered indicative of severe multicollinearity [16, 29].

The analysis is performed on three transformations of the data: raw price values (x_t), first differences ($\Delta x_t = x_t - x_{t-1}$), and absolute first differences ($|\Delta x_t| = |x_t - x_{t-1}|$). The rationale for analyzing absolute first differences is based on the observation that in many markets, contracts behave antagonistically: if one contract increases, another often decreases. Thus, even when correlation in levels or differences is low or negative, co-movement—of any sign—can be detected via $|\Delta x_t|$.

The average statistics across markets are summarized in Table 5.1.

Statistic	**Raw Values**	**First Differences**	**Abs First Differences**
Mean Correlation	0.11	0.03	**0.22**
Median Correlation	0.19	0.03	**0.18**
Mean VIF	11110	1912	3220

Table 5.1: Average within-market correlation and VIF statistics across all markets.

These findings demonstrate that raw price levels already exhibit moderate correlation, while first differences show minimal correlation, likely due to transient noise. However, absolute first differences consistently reveal stronger co-movement, suggesting that contracts often move in opposing directions under mutual exclusivity constraints. Additionally, the extremely high VIF scores—sometimes approaching infinity—indicate severe multicollinearity across contracts in many markets. These empirical results invalidate the assumption of conditional independence and provide strong support for modeling contracts jointly via a shared latent state using the groupwise Beta-HMM.

5.2 Model Design

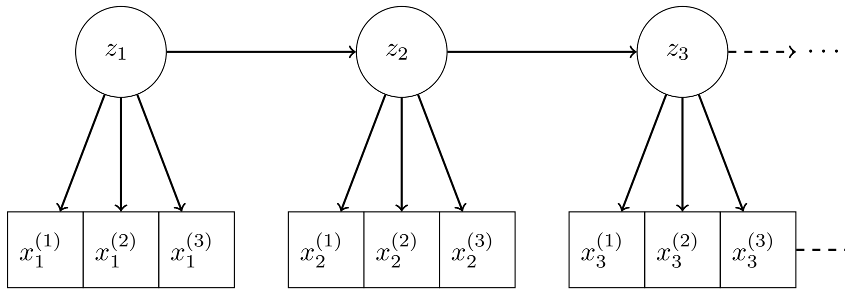

z_1
z_2
z_3
$\cdots$
$x_1^{(1)}$ $x_1^{(2)}$ $x_1^{(3)}$
$x_2^{(1)}$ $x_2^{(2)}$ $x_2^{(3)}$
$x_3^{(1)}$ $x_3^{(2)}$ $x_3^{(3)}$
$\cdots$

Figure 5.1: Graphical representation of the Groupwise Beta Hidden Markov Model (Groupwise Beta-HMM), where a shared hidden state z_t generates multiple observed emissions $x_t^{(d)}$ at each time step.

The standard Beta Hidden Markov Model (Beta-HMM) applied earlier assumes that each contract evolves independently over time. For each contract c, the time series $\{x_t^{(c)}\}_{t=1}^T$ is modeled as emissions from a latent Markov chain $\{z_t^{(c)}\}$, with each hidden state $z_t^{(c)} \in \{1, \ldots, K\}$ having contract-specific emission parameters $\alpha_k^{(c)}, \beta_k^{(c)}$, governing the Beta distribution:

$$x_t^{(c)} \sim \text{Beta}(\alpha_{z_t^{(c)}}^{(c)}, \beta_{z_t^{(c)}}^{(c)}).$$

This approach treats contracts as *independent*, neglecting interdependencies that arise in grouped or mutually exclusive settings—such as candidate contracts in an election market where only one can resolve to 1.

To incorporate this mutual exclusivity, This paper proposes a new structured Beta-HMM variant where multiple contracts are modeled jointly within a group. Let a group consist of D contracts. Instead of modeling D independent HMMs, this paper defines a single shared latent state $z_t \in \{0, 1, \ldots, D\}$ at time t with the following interpretation:

- $z_t = 0$: No dominant contract — general trading behavior.
- $z_t = k \in \{1, \ldots, D\}$: Contract k is dominant, i.e., expected to resolve to 1.

This shared state z_t controls *emission distributions for all contracts jointly.* Let $\mathbf{x}_t \in [0, 1]^D$ denote the observed vector of contract prices at time t. Then:

$$\mathbf{x}_t \sim \prod_{d=1}^{D} \mathrm{Beta}(\alpha_{z_t,d}, \beta_{z_t,d}),$$

where the parameters $\alpha_{z_t,d}, \beta_{z_t,d}$ are defined deterministically based on the interpretation of z_t:

5.3 Groupwise Emissions and Masked Training in Heterogeneous Prediction Markets

A central innovation of the Groupwise Beta-HMM is its ability to handle structurally heterogeneous prediction markets—i.e., markets that vary in the number and identity of outcome contracts—within a unified probabilistic model. Classical HMMs assume a fixed-dimensional observation space across all sequences, an assumption violated in our setting where each market has a unique contract composition [39]. To address this, all markets are embedded into a common observation space, and a masked emission framework is employed to maintain computational consistency while respecting market-specific features.

Let $\mathcal{C} = \{c_1, \ldots, c_D\}$ denote the global set of unique contracts across all markets. Each market i yields a trajectory $\{\mathbf{x}_t^{(i)}\}_{t=1}^{T_i}$ with $\mathbf{x}_t^{(i)} \in [0, 1]^D$, where unobserved contracts are omitted via a binary mask $\mathbf{m}_t^{(i)} \in \{0, 1\}^D$ that indicates the active dimensions. This mask is used throughout the EM algorithm to ensure that only available data contribute to likelihood computation, posterior inference, and parameter estimation.

Each hidden state $z_t \in \{0, 1, \ldots, D\}$ represents a latent market regime. State $z_t = 0$ corresponds to a non-dominant, uncertain regime where all contracts follow a neutral prior. States $z_t = k \in \{1, \ldots, D\}$ indicate dominance of contract c_k. Emissions are modeled as independent Beta distributions per contract:

$$p(\mathbf{x}_t^{(i)} \mid z_t = k) = \prod_{d=1}^{D} \mathrm{Beta}(x_{t,d}^{(i)}; \alpha_{k,d}, \beta_{k,d})^{m_{t,d}^{(i)}},$$

where the product is taken only over observed dimensions. The corresponding log-likelihood becomes:

$$\log p(\mathbf{x}_t^{(i)} \mid z_t = k) = \sum_{d=1}^{D} m_{t,d}^{(i)} \log \mathrm{Beta}(x_{t,d}^{(i)}; \alpha_{k,d}, \beta_{k,d}).$$

Emission parameters are fixed a priori. In the neutral regime $z_t = 0$, all contracts follow $\mathrm{Beta}(2, 2)$, centered at 0.5. In dominant regimes $z_t = k \geq 1$, contract $d = k$ follows $\mathrm{Beta}(30, 2)$, concentrating near 1, while all other contracts $d \neq k$ follow $\mathrm{Beta}(2, 30)$, concentrating near 0. This induces a soft mutual exclusivity consistent with prediction market structure, where only one contract resolves to 1.

State transitions follow a first-order Markov process with transition matrix $A \in \mathbb{R}^{(D+1)\times(D+1)}$ and initial state distribution $\delta \in \mathbb{R}^{D+1}$. These parameters are learned via the Expectation-Maximization (EM) algorithm using masked operations in both E-step and M-step.

In the E-step, the forward and backward probabilities are computed using masked emissions, ensuring that only observed contract values contribute to each recursion. Let $\alpha_t^{(i)}(k)$ and $\beta_t^{(i)}(k)$ denote the forward and backward variables, respectively. The masked emission likelihood $f(\mathbf{x}_t^{(i)} \mid z_t = k)$ is inserted into the standard recursions:

$$\alpha_t^{(i)}(k) = \sum_{j=0}^{D} \alpha_{t-1}^{(i)}(j) A_{jk} f(\mathbf{x}_t^{(i)} \mid z_t = k),$$

$$\beta_t^{(i)}(k) = \sum_{j=0}^{D} A_{kj} f(\mathbf{x}_{t+1}^{(i)} \mid z_{t+1} = j) \beta_{t+1}^{(i)}(j).$$

Posterior state probabilities are computed as:

$$\gamma_t^{(i)}(k) = \frac{\alpha_t^{(i)}(k)\beta_t^{(i)}(k)}{\sum_{l=0}^{D} \alpha_t^{(i)}(l)\beta_t^{(i)}(l)}.$$

In the M-step, expected sufficient statistics from the E-step are used to update the transition matrix A and initial distribution δ. For transition updates, the joint posterior $\xi_t^{(i)}(j, k)$ is computed using masked likelihoods:

$$\xi_t^{(i)}(j, k) = \frac{\alpha_t^{(i)}(j) A_{jk} f(\mathbf{x}_{t+1}^{(i)} \mid z_{t+1} = k)\beta_{t+1}^{(i)}(k)}{\sum_{u=0}^{D} \sum_{v=0}^{D} \alpha_t^{(i)}(u) A_{uv} f(\mathbf{x}_{t+1}^{(i)} \mid z_{t+1} = v)\beta_{t+1}^{(i)}(v)}.$$

The updated transition probabilities are:

$$A_{jk} = \frac{\sum_i \sum_{t=1}^{T_i - 1} \xi_t^{(i)}(j, k)}{\sum_i \sum_{t=1}^{T_i - 1} \sum_{v=0}^{D} \xi_t^{(i)}(j, v)}, \qquad \delta_k = \frac{\sum_i \gamma_1^{(i)}(k)}{\sum_i 1}.$$

This architecture enables pooled training across all markets while ensuring that only observed contract values contribute to parameter estimation. The masking framework ensures that each emission density is computed correctly and consistently, despite missing data and variable contract sets. This approach integrates ideas from masked probabilistic inference [5], parameter tying under heterogeneous views [22], and collaborative sequence modeling [32], enabling scalable learning in structured but incomplete prediction market environments.

5.4 Masked Viterbi Decoding for Structured Prediction

Building on the masked emission framework used during EM training, the Viterbi algorithm must also be adapted to ensure consistency in inference. In the classical HMM setting, decoding involves computing the most probable sequence of hidden states $z_1^*, z_2^*, \ldots, z_T^*$ given the observed data. However, in the groupwise model with variable observation structures across markets, standard Viterbi recursion cannot be directly applied. Instead, the masked emission likelihoods introduced earlier must be used during the dynamic programming recursion to respect missing dimensions.

Let $\mathbf{x}_t \in [0,1]^D$ denote the padded observation at time t and $\mathbf{m}_t \in \{0,1\}^D$ its corresponding binary mask. The log-emission probability for state $z_t = k$ is given by:

$$\log p(\mathbf{x}_t \mid z_t = k) = \sum_{d=1}^{D} m_{t,d} \log \text{Beta}(x_{t,d}; \alpha_{k,d}, \beta_{k,d}),$$

as in the EM algorithm. This ensures that only observed dimensions contribute to the decoding score.

Let $\delta_t(k)$ denote the maximum log-probability of any path ending in state k at time t, and let $\psi_t(k)$ store the corresponding backpointer. The recursion becomes:

$$\delta_t(k) = \max_j \left[\delta_{t-1}(j) + \log A_{jk}\right] + \log p(\mathbf{x}_t \mid z_t = k),$$

$$\psi_t(k) = \arg\max_j \left[\delta_{t-1}(j) + \log A_{jk}\right],$$

where the emission term uses the masked log-likelihood above.

This masked Viterbi decoding procedure maintains consistency with the training regime and enables accurate state path reconstruction even when contracts are partially observed. Importantly, it supports structured prediction under heterogeneity, generalizing standard Viterbi decoding to real-world domains with incomplete or asymmetric information.

The use of masking during decoding follows principles of partially observed sequence modeling and structured dynamic programming under missing data [5, 39, 44]. This ensures that inference remains robust and probabilistically coherent in the presence of structural gaps.

5.5 Comparison to the Standard Beta-Hidden Markov Model

The Groupwise Beta-HMM differs from the standard Beta-HMM in several key aspects related to state structure, emission modeling, and training methodology. These modifications address core limitations in modeling prediction markets, particularly the need to capture joint outcome dependencies and accommodate structurally heterogeneous contract spaces.

Whereas the standard Beta-HMM models each contract in isolation, the groupwise formulation introduces a shared latent state space that captures market-level regime transitions. This design is essential in prediction markets like Polymarket, where contracts are mutually exclusive and interact competitively. The use of fixed Beta priors for emissions further encodes domain knowledge about expected behavior—such as the unimodal convergence of one contract toward 1 and others toward 0—without requiring extensive parameter estimation.

Feature	**Standard Beta-HMM**	**Groupwise Beta-HMM (ours)**
Observation space	Fixed-dimensional across markets	Variable (masked embedding in global space)
Contracts modeled	Independently (1 HMM per contract)	Jointly (shared hidden state)
Hidden state structure	Per-contract latent sequence	Shared latent regime across group
Emission updates	Learned via method-of-moments (EM)	Structured Beta priors
Missing data handling	Not explicitly addressed	Masked emissions and inference
Outcome structure	Arbitrary across contracts	Mutually exclusive competition
Number of states	Fixed per contract	$D + 1$ per group
Decoding method	Standard Viterbi	Masked Viterbi over heterogeneous space

Table 5.2: Comparison between the Standard and Groupwise Beta-HMM frameworks.

Crucially, the groupwise model generalizes to structurally heterogeneous settings via masked emission training and decoding. This ensures consistency in statistical inference even when the number and identity of contracts vary across markets. The result is a scalable, interpretable, and statistically coherent framework for modeling real-world prediction markets, particularly those with sparse, asymmetric, or variable market structure [22, 32, 5].

Chapter 6

Performance Evaluation

To evaluate the effectiveness of the modeling approaches, this chapter compares three models along two key dimensions: statistical accuracy and economic profitability. Accuracy measures the proportion of correct outcome predictions, while economic performance is captured through average per-trade profit and the timing of stable predictive signals.

The models considered are the naive benchmark, the Beta HMM, and the Groupwise Beta HMM. The naive benchmark is a rule-based strategy that takes a position five hours before resolution based on the higher observed market price. This model includes no statistical inference and serves as a simple baseline. The Beta HMM uses Beta-distributed emissions to accommodate the bounded nature of prediction market prices, improving fit and inference over traditional Gaussian-based approaches [43]. The Groupwise Beta HMM builds on this by jointly modeling outcome contracts with shared hidden states and masked training, reflecting interdependencies across outcomes and improving estimation under heterogeneous market structures [5, 32].

Each model is evaluated on a test set of 200 resolved markets using four key metrics. Classification accuracy refers to the share of predictions matching the resolved market outcome. Stable gap measures the average number of hours between the model's first stable prediction and final resolution, representing how early actionable information becomes available. Price difference is evaluated separately for correct and incorrect predictions, reflecting the margin of confidence in each regime. Finally, average profit measures the mean realized gain per trade, computed as the difference between entry and resolution prices.

6.1 Performance Comparison

Model	**Accuracy**	**Stable Gap (hrs)**	**Price Diff (Y/N)**	**Avg Profit ($)**
Naive Benchmark	79.8%	8.00	0.14 / 0.44	0.023
Beta HMM	86.9%	10.38	0.15 / 0.32	0.088
Groupwise Beta HMM	**89.3%**	**10.52**	**0.15 / 0.31**	**0.101**

Table 6.1: Statistical and Economic Performance Across Models

6.2 Discussion

Each successive model yields measurable improvements in both predictive accuracy and trading profitability. The Beta HMM improves upon the naive benchmark by incorporating time-series structure and probabilistic reasoning, reducing the average price difference in incorrect predictions from 0.44 to 0.32 and increasing profits nearly fourfold. The Groupwise Beta HMM further improves prediction accuracy to 89.3%, reduces error margins for incorrect trades, and yields the highest average profit per trade at $0.101. Additionally, its longer stable gap of 10.52 hours enables earlier trading signals, which is critical in fast-moving markets.

These findings validate the modeling choices made in each extension. The use of bounded emissions addresses the support mismatch in Gaussian-based models, while structured hidden states and groupwise masking align the model architecture with the true market design. The Groupwise Beta HMM, in particular, offers a practical and statistically grounded framework for real-time forecasting and decision-making in decentralized prediction markets.

Chapter 7

Conclusion

This thesis investigated the informational efficiency of decentralized prediction markets using advanced time series models on Polymarket data. Starting from a naive benchmark and progressing through a Gaussian HMM, a Beta-HMM, and finally a Groupwise Beta-HMM, each model incorporated increasing structural insight into market behavior. Results demonstrated that embedding market constraints—such as bounded probabilities and interdependent contracts—significantly improves prediction accuracy and profitability.

The Beta-HMM achieved 86.91% classification accuracy and \$0.0927 profit per trade, outperforming the naive benchmark's 72.8% and \$0.061. The Groupwise Beta-HMM reached 89.3% accuracy and \$0.110 per trade, confirming the advantage of modeling cross-contract dynamics.

Additionally, a convergence time statistic τ_ε was introduced to quantify belief stabilization. Results showed markets resolving to "No" converge more rapidly than those resolving to "Yes," reflecting asymmetric information dynamics and possible behavioral biases [47, 4].

These findings highlight the practical value of probabilistic modeling for real-time inference in speculative environments and underscore the role of prediction markets as natural laboratories for studying belief formation.

7.1 Economic Implications and Feasibility

The proposed trading strategy based on the Groupwise Beta-HMM yields an average profit of \$0.101 per trade over a 10.52-hour window, implying strong short-term returns under idealized conditions. However, translating this performance into live trading requires accounting for several critical frictions.

First, transaction costs—such as platform fees, bid-ask spreads, and gas fees on blockchain-based platforms—can materially reduce net profitability. For instance, on Polymarket, typical spreads can exceed 1–2 cents per contract, which, when combined with slippage in volatile or illiquid markets, may offset a significant portion of expected gains. These frictions are consistent with the findings in empirical microstructure studies on decentralized exchanges [27, 24]. Accurate cost modeling should be incorporated into future backtests.

Second, the backtest conducted in this study assumes immediate and frictionless execution at quoted mid-prices. This assumption is optimistic. A cleaner, event-driven backtest architecture is needed to reflect realistic latency, order book depth, and partial fill risks. Robust backtesting frameworks such as those described in [3] emphasize the importance of accounting for execution constraints and market impact. In practice, order placement algorithms would need to balance fill probability against adverse selection risk, particularly when convergence signals arise close to resolution time.

Third, reliable real-time data access is a prerequisite for implementing the strategy in production. While Polymarket publishes live data feeds, latency, outages, or synchronization lags could impair timely signal detection and execution.

Finally, regulatory feasibility remains a substantial constraint. In jurisdictions such as the United States, trading on unregulated event contracts may violate securities or derivatives laws. Polymarket itself was fined by the U.S. Commodity Futures Trading Commission (CFTC) in 2021 and required to cease offering certain contracts to U.S. persons [10]. As such, implementation would require careful legal review and may only be feasible for offshore entities or non-U.S. residents.

Despite these constraints, the strategy remains conceptually viable in environments with low transaction costs, sufficient liquidity, and clear legal status. The performance metrics suggest that, even after modest cost deductions, profitable execution may be possible in selected markets.

7.2 Outlook and Future Work

The groupwise modeling framework can be extended in several directions. Future research could explore hierarchical HMMs or mixed-membership models that allow for overlapping latent regimes [6]. Attention-based methods such as Transformers [46] may also offer a more flexible way to model inter-contract dependencies.

Another promising direction is domain stratification: applying the model separately to thematic subsets (e.g., politics, sports, finance) may uncover domain-specific convergence dynamics and improve predictive precision.

Finally, incorporating external signals—such as breaking news or social media indicators—into the emission layer would allow conditioning on exogenous information, paving the way for richer and more responsive prediction models [34, 43].

7.3 Disclaimer

This paper is intended for academic and educational purposes only. It does not constitute investment advice or an endorsement of any trading strategy. Readers should be aware that trading on Polymarket is currently restricted for users in the United States. Any discussion of potential profitability is hypothetical and should not be interpreted as financial guidance or an invitation to circumvent legal restrictions.

7.4 Appendices: Code

7.4.1 Naive Benchmark Model

The following Python code contains the implementation of the Naive Benchmark Model as well as the preceding data manipulation.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

class NaiveBenchmark:
    def __init__(self):
        pass

    def fit(self, X):
        pass  # No training needed

    def predict(self, X):
        return np.where(X > 0.5, 1, 0)

# Load and prepare data
file_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\6th Sem
    Stetson\second semester time series\data\data_withcat_cleantails_condensed.csv"
data = pd.read_csv(file_path, index_col=0)

# Extract outcome and prepare data
outcome_yes = data.loc["outcome_yes"].astype(int)
data = data.drop(index="outcome_yes")
data = data.dropna(axis=0, how='any')
constant_cols = [col for col in data.columns if np.all(data[col].values == data[col
    ].values[0])]
data = data.drop(columns=constant_cols)
outcome_yes = outcome_yes.drop(labels=constant_cols)

# Track multiple runs
num_runs = 10
results = []

for run in range(num_runs):
    # Split data
    train_cols, test_cols = train_test_split(data.columns, test_size=0.2,
    random_state=run)
    train_data = data[train_cols]
    test_data = data[test_cols]
    train_outcomes = outcome_yes[train_cols]
    test_outcomes = outcome_yes[test_cols]

    # Train Naive Benchmark model
    model = NaiveBenchmark()
    model.fit(None)  # No actual training

    # Evaluation
    correct_predictions = 0
    total_predictions = 0
    total_price_diff = 0
    price_diff_correct_sum = 0
    price_diff_incorrect_sum = 0
    count_correct = 0
    count_incorrect = 0

    for col in test_data.columns:
        series = test_data[col].values
        true_outcome = int(test_outcomes[col])

        if len(series) > 8:
            predicted_outcome = model.predict(series[-3])  # Use the observation 10 steps before the last one
        else:
            predicted_outcome = model.predict(series[0])  # If not enough data, use the first observation

        price_resolving = series[-1]
        price_diff = abs(1 - price_resolving) if predicted_outcome == 1 else abs(0 - price_resolving)

        total_predictions += 1
        total_price_diff += price_diff

        if predicted_outcome == true_outcome:
            correct_predictions += 1
            price_diff_correct_sum += price_diff
            count_correct += 1
        else:
            price_diff_incorrect_sum += price_diff
            count_incorrect += 1

    # Calculate and store results
    if total_predictions > 0:
        accuracy = (correct_predictions / total_predictions) * 100
        avg_price_diff = total_price_diff / total_predictions
        avg_price_diff_correct = price_diff_correct_sum / count_correct if count_correct > 0 else 0
        avg_price_diff_incorrect = price_diff_incorrect_sum / count_incorrect if count_incorrect > 0 else 0
        profit = (correct_predictions / total_predictions) * avg_price_diff_correct - (1 - (correct_predictions / total_predictions)) * avg_price_diff_incorrect

        results.append([run + 1, accuracy, avg_price_diff, avg_price_diff_correct, avg_price_diff_incorrect, profit])
    else:
        results.append([run + 1, None, None, None, None, None])

# Print results
print("\n========== Summary of 10 Runs ==========")
print(f"{'Run':<5} {'Accuracy (%)':<15} {'Price Diff':<15} {'Price Diff (Correct)':<22} {'Price Diff (Incorrect)':<22} {'Profit':<10}")
print("=" * 90)
for res in results:
    run_num, acc, price_diff, price_diff_correct, price_diff_incorrect, profit = res
    if acc is not None:
        print(f"{run_num:<5} {acc:<15.2f} {price_diff:<15.2f} {price_diff_correct:<22.2f} {price_diff_incorrect:<22.2f} {profit:<10.4f}")
    else:
        print(f"{run_num:<5} {'N/A':<15} {'N/A':<15} {'N/A':<22} {'N/A':<22} {'N/A':<10}")
```

```python

# Calculate and print averages
valid_results = [res[1:] for res in results if res[1] is not None]
if valid_results:
    avg_results = np.mean(valid_results, axis=0)
    print("\n========== Averages Across Runs ==========")
    print(f"Average Accuracy: {avg_results[0]:.2f}%")
    print(f"Average Price Difference: {avg_results[1]:.2f}")
    print(f"Average Price Difference (Correct Predictions): {avg_results[2]:.2f}")
    print(f"Average Price Difference (Incorrect Predictions): {avg_results[3]:.2f}"
    )
    print(f"Average Profit: {avg_results[4]:.4f}")
```

Listing 7.1: Modified LMSR Implementation

7.4.2 Basic Beta Hidden Markov Model

The following Python code contains the implementation of the Basic Beta Hidden Markov Model as well as the preceding data manipulation.

```python
import pandas as pd
import numpy as np
from scipy.stats import beta
from sklearn.model_selection import train_test_split

class BetaHMM:
    def __init__(self, n_components=3, n_iter=100, tol=1e-3, random_state=None):
        self.n_components = n_components
        self.n_iter = n_iter
        self.tol = tol
        self.random_state = random_state
        np.random.seed(random_state)

    def _initialize(self, X):
        # Initialize parameters (vectorized)
        self.transmat_ = np.random.rand(self.n_components, self.n_components)
        self.transmat_ /= self.transmat_.sum(axis=1)[:, np.newaxis]

        self.startprob_ = np.random.rand(self.n_components)
        self.startprob_ /= self.startprob_.sum()

        # Pre-compute beta parameters for different scenarios
        self.alpha_ = np.array([1.0, 5.0, 1.0])
        self.beta_ = np.array([5.0, 1.0, 1.0])

    def _emission_prob_matrix(self, X):
        # Vectorized emission probability calculation
        return np.array([beta.pdf(X, a, b) for a, b in zip(self.alpha_, self.beta_)
    ]).T

    def fit(self, X):
        self._initialize(X)
        X = np.clip(X, 0.001, 0.999)  # Avoid boundary issues

        for _ in range(self.n_iter):
            # Pre-compute emission probabilities for all observations
            emit_prob = self._emission_prob_matrix(X)

            # Forward pass (vectorized)
            forward = np.zeros((len(X), self.n_components))
            forward[0] = self.startprob_ * emit_prob[0]

            for t in range(1, len(X)):
                forward[t] = emit_prob[t] * np.dot(forward[t-1], self.transmat_)

            # Backward pass (vectorized)
            backward = np.zeros((len(X), self.n_components))
            backward[-1] = 1

            for t in range(len(X)-2, -1, -1):
                backward[t] = np.dot(self.transmat_, (emit_prob[t+1] * backward[t
    +1]))

```

```
            # Calculate posteriors (vectorized)
            posteriors = forward * backward
            posteriors /= posteriors.sum(axis=1)[:, np.newaxis]

            # Update parameters (vectorized)
            # Transition matrix update
            xi_sum = np.zeros((self.n_components, self.n_components))
            for t in range(len(X)-1):
                xi_sum += np.outer(forward[t], backward[t+1] * emit_prob[t+1]) *
    self.transmat_
            self.transmat_ = xi_sum / xi_sum.sum(axis=1)[:, np.newaxis]

            # Emission parameters update (vectorized)
            for k in range(self.n_components):
                weights = posteriors[:, k]
                if weights.sum() > 0:
                    weighted_x = X * weights
                    mean = weighted_x.sum() / weights.sum()
                    var = np.sum(weights * (X - mean)**2) / weights.sum()

                    # Method of moments with bounds
                    if 0 < mean < 1 and var > 0:
                        mean = np.clip(mean, 0.01, 0.99)
                        var = np.clip(var, 0.001, 0.25)

                        common_factor = mean * (1 - mean) / var - 1
                        self.alpha_[k] = mean * common_factor
                        self.beta_[k] = (1 - mean) * common_factor

        return self

    def predict(self, X):
        X = np.clip(X, 0.001, 0.999)
        emit_prob = self._emission_prob_matrix(X)

        # Forward pass only (vectorized)
        forward = np.zeros((len(X), self.n_components))
        forward[0] = self.startprob_ * emit_prob[0]

        for t in range(1, len(X)):
            forward[t] = emit_prob[t] * np.dot(forward[t-1], self.transmat_)

        return np.argmax(forward, axis=1)


# Load and prepare data (same as original)
file_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\6th Sem
    Stetson\second semester time series\data\
    data_withcat_cleantails_condensed_multimarket.csv"
data = pd.read_csv(file_path, index_col=0)

# Extract outcome and prepare data (same as original)
outcome_yes = data.loc["outcome_yes"].astype(int)
data = data.drop(index="outcome_yes")
data = data.dropna(axis=0, how='any')
constant_cols = [col for col in data.columns if np.all(data[col].values == data[col
    ].values[0])]
data = data.drop(columns=constant_cols)
outcome_yes = outcome_yes.drop(labels=constant_cols)
```

```python

# Track multiple runs
num_runs = 10
results = []

for run in range(num_runs):
    # Split data (same as original)
    train_cols, test_cols = train_test_split(data.columns, test_size=0.2,
    random_state=run)
    train_data = data[train_cols]
    test_data = data[test_cols]
    train_outcomes = outcome_yes[train_cols]
    test_outcomes = outcome_yes[test_cols]

    # Prepare sequences
    all_train_sequences = [train_data[col].values.reshape(-1, 1) for col in
    train_data.columns]
    train_sequences = np.concatenate(all_train_sequences, axis=0)

    # Train the Beta HMM
    model = BetaHMM(n_components=3, n_iter=100, random_state=run)
    model.fit(train_sequences.flatten())  # Beta HMM expects 1D array

    # Evaluation (same logic as original)
    correct_predictions = 0
    total_predictions = 0
    total_stable_gap = 0
    total_price_diff = 0
    price_diff_correct_sum = 0
    price_diff_incorrect_sum = 0
    count_correct = 0
    count_incorrect = 0
    errors = []

    for col in test_data.columns:
        series = test_data[col].values
        true_outcome = int(test_outcomes[col])
        try:
            hidden_states = model.predict(series)
        except Exception as e:
            errors.append(f"Error in column {col}: {e}")
            continue

        final_state = hidden_states[-1]
        stable_index = next((i + 1 for i in range(len(hidden_states) - 1, -1, -1)
    if hidden_states[i] != final_state), 0)
        stable_gap = (len(hidden_states) - 1) - stable_index
        price_resolving = series[stable_index]
        predicted_outcome = 1 if final_state == 1 else (0 if final_state == 2 else
    (1 if series[-1] > 0.5 else 0))
        price_diff = abs(1 - price_resolving) if predicted_outcome == 1 else abs(0
    - price_resolving)

        total_stable_gap += stable_gap
        total_price_diff += price_diff
        total_predictions += 1

        if predicted_outcome == true_outcome:
            correct_predictions += 1
            price_diff_correct_sum += price_diff
            count_correct += 1
        else:
            price_diff_incorrect_sum += price_diff
            count_incorrect += 1

    # Calculate and store results (same as original)
    if total_predictions > 0:
        accuracy = (correct_predictions / total_predictions) * 100
        avg_stable_gap = total_stable_gap / total_predictions
        avg_price_diff = total_price_diff / total_predictions
        avg_price_diff_correct = price_diff_correct_sum / count_correct if count_correct > 0 else 0
        avg_price_diff_incorrect = price_diff_incorrect_sum / count_incorrect if count_incorrect > 0 else 0
        profit = (correct_predictions / total_predictions) * avg_price_diff_correct - (1 - (correct_predictions / total_predictions)) * avg_price_diff_incorrect

        results.append([run + 1, accuracy, avg_stable_gap, avg_price_diff, avg_price_diff_correct, avg_price_diff_incorrect, profit])
    else:
        results.append([run + 1, None, None, None, None, None, None])

# Print results (same as original)
print("\n========== Summary of 10 Runs ==========")
print(f"{'Run':<5} {'Accuracy (%)':<15} {'Stable Gap':<15} {'Price Diff':<15} {'Price Diff (Correct)':<22} {'Price Diff (Incorrect)':<22} {'Profit':<10}")
print("=" * 110)
for res in results:
    run_num, acc, stable_gap, price_diff, price_diff_correct, price_diff_incorrect, profit = res
    if acc is not None:
        print(f"{run_num:<5} {acc:<15.2f} {stable_gap:<15.2f} {price_diff:<15.2f} {price_diff_correct:<22.2f} {price_diff_incorrect:<22.2f} {profit:<10.4f}")
    else:
        print(f"{run_num:<5} {'N/A':<15} {'N/A':<15} {'N/A':<15} {'N/A':<22} {'N/A':<22} {'N/A':<10}")

# Calculate and print averages (same as original)
valid_results = [res[1:] for res in results if res[1] is not None]
if valid_results:
    avg_results = np.mean(valid_results, axis=0)
    print("\n========== Averages Across Runs ==========")
    print(f"Average Accuracy: {avg_results[0]:.2f}%")
    print(f"Average Stable Gap: {avg_results[1]:.2f}")
    print(f"Average Price Difference: {avg_results[2]:.2f}")
    print(f"Average Price Difference (Correct Predictions): {avg_results[3]:.2f}")
    print(f"Average Price Difference (Incorrect Predictions): {avg_results[4]:.2f}")
    print(f"Average Profit: {avg_results[5]:.4f}")
```
Listing 7.2: Modified Sigmoid Implementation

7.4.3 Groupwise Beta Hidden Markov Model

The following Python code contains the implementation of the Groupwise Beta Hidden Markov Model as well as the preceding data manipulation.

```python
import os
import pandas as pd
import numpy as np
from scipy.stats import beta

###############################################################################
# 1) OneContractOrNoneDominantBetaHMM Class with  M e t h o d ofMoments  Emission
    Updates
###############################################################################
class OneContractOrNoneDominantBetaHMM:
    """
    HMM with K = D + 1 states, where D is the number of contracts.
      - State 0 ("none dominant"): initially, parameters may be near neutral (e.g.
    Beta(2,2)),
        but will be estimated by method of moments.
      - States k (for k = 1,...,D): in state k, the emission parameters for each
    contract are
        estimated via method of moments from the data.
    In this implementation the emission parameters (alpha and beta for each state
    and contract)
    are updated during each EM iteration using a weighted method of moments.
    Only the transition matrix and start probabilities are also learned via EM.
    """
    def __init__(self, n_iter=50, tol=1e-3, random_state=None):
        self.n_iter = n_iter
        self.tol = tol
        self.random_state = random_state
        if random_state is not None:
            np.random.seed(random_state)

    def _initialize_params(self, X):
        # X: shape (T, D) where T = number of time steps, D = number of contracts.
        T, D = X.shape
        self.D = D
        self.K = D + 1  # one extra state for "none dominant"

        # Randomly initialize transition matrix and start probabilities.
        self.transmat_ = np.random.rand(self.K, self.K)
        self.transmat_ /= self.transmat_.sum(axis=1, keepdims=True)

        self.startprob_ = np.random.rand(self.K)
        self.startprob_ /= self.startprob_.sum()

        # Initialize emission parameters with reasonable starting values.
        # For example, start with Beta(2,2) for state 0 and Beta(2,30) for non-
    dominant contracts,
        # Beta(30,2) for the contract that is "dominant" in each state.
        self.alpha_ = np.zeros((self.K, self.D))
        self.beta_  = np.zeros((self.K, self.D))

        # For state 0 ("none dominant") initialize neutrally.
        for d in range(self.D):
            self.alpha_[0, d] = 2.0
            self.beta_[0, d]  = 2.0

        # For states 1...D, initialize such that contract (k-1) is dominant.
        for k in range(1, self.K):
            for d in range(self.D):
                if d == k - 1:
                    self.alpha_[k, d] = 30.0
                    self.beta_[k, d]  = 2.0
                else:
                    self.alpha_[k, d] = 2.0
                    self.beta_[k, d]  = 30.0

    def _compute_emission_probs(self, X):
        """
        For each time step t and state k, compute:
          p(x_t | z_t=k) =   _ {d=1}^{D} Beta(x[t,d]; alpha_[k,d], beta_[k,d])
        """
        T, D = X.shape
        emission_probs = np.zeros((T, self.K))
        for k in range(self.K):
            pdf_vals = beta.pdf(X, self.alpha_[k], self.beta_[k])  # shape (T, D)
            emission_probs[:, k] = np.prod(pdf_vals, axis=1)
        return emission_probs

    def fit(self, X):
        """
        Learn the transition matrix, start probabilities, and update emission parameters using EM.
        Emission parameters are updated via a weighted method-of-moments.
        """
        X = np.clip(X, 0.001, 0.999)
        T, D = X.shape
        self._initialize_params(X)

        for iteration in range(self.n_iter):
            # E-step: compute emission probabilities and forward-backward.
            emit_prob = self._compute_emission_probs(X)  # shape (T, K)

            # Forward pass.
            forward = np.zeros((T, self.K))
            forward[0] = self.startprob_ * emit_prob[0]
            for t in range(1, T):
                forward[t] = emit_prob[t] * np.dot(forward[t-1], self.transmat_)

            # Backward pass.
            backward = np.zeros((T, self.K))
            backward[-1] = 1.0
            for t in range(T-2, -1, -1):
                backward[t] = np.dot(self.transmat_, (emit_prob[t+1] * backward[t+1]))

            gamma = forward * backward
            gamma /= np.where(gamma.sum(axis=1, keepdims=True) > 0,
                              gamma.sum(axis=1, keepdims=True), 1e-12)

            # M-step: update transition and start probabilities.
            xi_sum = np.zeros((self.K, self.K))
            for t in range(T - 1):
                temp = np.outer(forward[t], emit_prob[t+1] * backward[t+1])
                xi_sum += temp * self.transmat_
            row_sums = xi_sum.sum(axis=1, keepdims=True)
            row_sums[row_sums == 0] = 1e-12
            self.transmat_ = xi_sum / row_sums

            self.startprob_ = gamma[0] / gamma[0].sum()

            # M-step: update emission parameters via weighted method-of-moments.
            # For each state k and each contract dimension d:
            for k in range(self.K):
                weights = gamma[:, k]  # shape (T,)
                weight_sum = weights.sum()
                if weight_sum > 0:
                    for d in range(self.D):
                        # Compute weighted mean and variance for X[:, d].
                        m = np.sum(weights * X[:, d]) / weight_sum
                        var = np.sum(weights * (X[:, d] - m) ** 2) / weight_sum
                        # Clip values to avoid extreme parameters.
                        m = np.clip(m, 0.01, 0.99)
                        var = np.clip(var, 1e-4, 0.25)
                        common_factor = m * (1 - m) / var - 1
                        # Avoid division by zero or negative values.
                        if common_factor > 0:
                            self.alpha_[k, d] = m * common_factor
                            self.beta_[k, d] = (1 - m) * common_factor
                        else:
                            # If common_factor is not positive, keep previous values.
                            pass
            # (Optionally, you could compute the log-likelihood here and check for convergence.)
        return self

    def predict_states(self, X):
        """
        Predict the most likely state at each time step using a forward pass (via argmax).
        """
        X = np.clip(X, 0.001, 0.999)
        T, D = X.shape
        emit_prob = self._compute_emission_probs(X)
        forward = np.zeros((T, self.K))
        forward[0] = self.startprob_ * emit_prob[0]
        for t in range(1, T):
            forward[t] = emit_prob[t] * np.dot(forward[t-1], self.transmat_)
        return np.argmax(forward, axis=1)

###############################################################################
# 2) Main: Process all market files from folder (groupwise evaluation)
###############################################################################
def main():
    # Folder containing separate market files.
    folder_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\6th Sem Stetson\second semester time series\data\processed_outcome"

    # List all CSV files in the folder.
    file_list = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.endswith('.csv')]
    if not file_list:
        raise ValueError("No CSV files found in the folder.")

    print(f"Found {len(file_list)} market files.")

    # Overall accumulators for evaluation metrics.
    total_correct = 0
    total_predictions = 0
    total_stable_gap = 0
    total_price_diff = 0
    total_price_diff_correct = 0
    total_price_diff_incorrect = 0
    total_count_correct = 0
    total_count_incorrect = 0
    num_markets_evaluated = 0

    # Process each market file.
    for file_path in file_list:
        print(f"\nProcessing file: {file_path}")
        try:
            df = pd.read_csv(file_path, header=0, index_col=0, low_memory=False)
        except Exception as e:
            print(f"Error reading {file_path}: {e}")
            continue

        print("File shape:", df.shape)
        # Expected structure:
        # - Header row with contract names.
        # - Row with index "outcome_yes": final outcomes for each contract.
        # - Remaining rows (indexed by time steps) are the time series.
        if "outcome_yes" not in df.index:
            print("File skipped (no 'outcome_yes' row).")
            continue

        # Extract outcome row and convert to numeric.
        outcome_row = df.loc["outcome_yes"].copy().apply(lambda x: pd.to_numeric(x,
 errors='coerce'))

        # Get the time series data (all rows except outcome_yes).
        ts_df = df.drop(index="outcome_yes").copy()
        try:
            ts_df.index = pd.to_numeric(ts_df.index)
        except Exception:
            pass

        # Convert all contract columns to numeric.
        contract_cols = list(df.columns)
        ts_df[contract_cols] = ts_df[contract_cols].apply(pd.to_numeric, errors='
coerce')
        ts_df = ts_df.dropna(axis=0, how='any')

        print("Contract columns in file:", contract_cols)

        # Skip the market if no contract has outcome 1.
        if outcome_row.sum() == 0:
            print("No contract outcome equals 1; skipping file.")
            continue

        # Use the entire time series (all time steps) for training/testing.
        X = ts_df[contract_cols].values  # shape (T, D)
```

```python
    if X.shape[0] < 2:
        print("Not enough time-series data; skipping file.")
        continue

    # Train the model on the full time series of this market.
    model = OneContractOrNoneDominantBetaHMM(n_iter=50, random_state=0)
    model.fit(X)

    # Predict hidden states on the full time series.
    hidden_states = model.predict_states(X)
    final_state = hidden_states[-1]

    # Determine the stable index: the first index from the end where the state changes.
    stable_index = len(hidden_states) - 1
    for i in range(len(hidden_states)-2, -1, -1):
        if hidden_states[i] != final_state:
            stable_index = i + 1
            break
    gap = (len(hidden_states) - 1) - stable_index
    total_stable_gap += gap

    # Get the prices at the stable time.
    stable_prices = X[stable_index, :]

    # Evaluate predictions for each contract.
    for idx, col in enumerate(contract_cols):
        try:
            true_outcome = int(outcome_row[col])
        except Exception:
            continue

        # Prediction logic:
        # If final_state == 0 ("none dominant"), predict 0 for all.
        # If final_state >= 1, then contract (final_state-1) is dominant predicted 1 for that contract, 0 for others.
        if final_state == 0:
            predicted_outcome = 0
        else:
            predicted_outcome = 1 if (final_state - 1) == idx else 0

        price_val = stable_prices[idx]
        diff = abs(1 - price_val) if predicted_outcome == 1 else abs(price_val - 0)

        total_predictions += 1
        total_price_diff += diff

        if predicted_outcome == true_outcome:
            total_correct += 1
            total_price_diff_correct += diff
            total_count_correct += 1
        else:
            total_price_diff_incorrect += diff
            total_count_incorrect += 1

    num_markets_evaluated += 1
    print(f"Market '{df.index[0]}' evaluated. Final state: {final_state}, Stable gap: {gap}")

    # Aggregate overall metrics.
    if total_predictions > 0:
        accuracy = (total_correct / total_predictions) * 100
        avg_gap = total_stable_gap / total_predictions
        avg_diff = total_price_diff / total_predictions
        avg_diff_correct = total_price_diff_correct / total_count_correct if total_count_correct > 0 else 0
        avg_diff_incorrect = total_price_diff_incorrect / total_count_incorrect if total_count_incorrect > 0 else 0
        frac_correct = total_correct / total_predictions
        profit = frac_correct * avg_diff_correct - (1 - frac_correct) * avg_diff_incorrect

        print("\n========== Aggregated Evaluation Metrics ==========")
        print(f"Number of markets evaluated: {num_markets_evaluated}")
        print(f"Total Predictions: {total_predictions}")
        print(f"Accuracy: {accuracy:.2f}%")
        print(f"Average Stable Gap (time steps): {avg_gap:.2f}")
        print(f"Average Price Difference: {avg_diff:.2f}")
        print(f"Average Price Diff (Correct): {avg_diff_correct:.2f}")
        print(f"Average Price Diff (Incorrect): {avg_diff_incorrect:.2f}")
        print(f"Profit: {profit:.4f}")
    else:
        print("No predictions were made across the market files.")

if __name__ == "__main__":
    main()
```

Listing 7.3: Exponential Smoothing Implementation

Bibliography

[1] Abramowitz, M., & Stegun, I. A. (1964). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* U.S. Government Printing Office.

[2] Arrow, K. J., Forsythe, R., Gorham, M., Hahn, R., Hanson, R., Ledyard, J. O., Levmore, S., List, J. A., Malebranche, E., Nelson, F. D., & Tetlock, P. C. (2008). The promise of prediction markets. *Science*, 320(5878), 877–878.

[3] Bailey, D. H., Borwein, J. M., Lopez de Prado, M., & Zhu, Q. J. (2014). The Probability of Backtest Overfitting. *Journal of Computational Finance*, 20(4), 39–69.

[4] Berg, J., Nelson, F., & Rietz, T. (2008). Prediction market accuracy in the long run. *International Journal of Forecasting*, 24(2), 285–300.

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

[6] Blei, D. M., & Lafferty, J. D. (2010). Mixed membership models for topic and attribute analysis. In *Proceedings of the 2010 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 79–88). ACM.

[7] Brand, M., Oliver, N., & Pentland, A. (1997). Coupled Hidden Markov Models for complex action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 994–999).

[8] Cappé, O., Moulines, E., & Rydén, T. (2005). *Inference in Hidden Markov Models.* Springer.

[9] Casella, G., & Berger, R. L. (2002). *Statistical Inference* (2nd ed.). Duxbury.

[10] U.S. Commodity Futures Trading Commission. (2022). CFTC Orders Polymarket to Pay $1.4 Million for Unlawfully Offering Event-Based Binary Options Contracts. Retrieved from `https://www.cftc.gov/PressRoom/PressReleases/8478-22`

[11] OpenAI (2025). ChatGPT (Mar 14 version) Model 4.0 https://chat.openai.com (used for proofreading, formulations, and latex code generation)

[12] Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1), 1-22.

[13] Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press.

[14] Durrett, R. (2012). *Essentials of Stochastic Processes* (2nd ed.). Springer.

[15] Durrett, R. (2019). *Probability: Theory and Examples* (5th ed.). Cambridge University Press.

[16] Farrar, D. E., & Glauber, R. R. (1967). Multicollinearity in regression analysis: The problem revisited. *The Review of Economics and Statistics*, 49(1), 92–107.

[17] Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE, 61*(3), 268–278.

[18] Galbraith, J. W. (2023). Alternative assets and portfolio diversification in volatile macroeconomic regimes. *Journal of Investment Strategies*, 12(1), 45–63.

[19] Gallager, R. G. (2013). *Stochastic Processes: Theory for Applications.* Cambridge University Press.

[20] Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis* (3rd ed.). CRC Press.

[21] Ghahramani, Z., & Jordan, M. I. (1996). Factorial Hidden Markov Models. In *Advances in Neural Information Processing Systems*, 8, 472–478.

[22] Ghahramani, Z., & Jordan, M. I. (1996). Parameter estimation for linear dynamical systems. Technical Report, Department of Cognitive and Brain Sciences, MIT.

[23] Grimmett, G., & Stirzaker, D. (2020). *Probability and Random Processes* (4th ed.). Oxford University Press.

[24] Hasbrouck, J. (2007). *Empirical Market Microstructure: The Institutions, Economics, and Econometrics of Securities Trading.* Oxford University Press.

[25] Hull, J. C. (2018). *Options, Futures, and Other Derivatives* (10th ed.). Pearson.

[26] Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous Univariate Distributions, Volume 2.* Wiley.

[27] Koeppel, A., Fricke, D., & Hackethal, A. (2023). DeFi Microstructure: Bid-Ask Spreads, Price Impact, and Liquidity on Uniswap. *Journal of Financial Markets*, 68, 100814.

[28] Kulkarni, V. G. (2010). *Modeling and Analysis of Stochastic Systems.* CRC Press.

[29] Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2004). *Applied Linear Statistical Models* (5th ed.). McGraw-Hill Irwin.

[30] Levin, D. A., Peres, Y., & Wilmer, E. L. (2009). *Markov Chains and Mixing Times.* American Mathematical Society.

[31] Marlin, B. M., & Zemel, R. S. (2007). Collaborative filtering on non-uniformly sampled data. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 267–274).

[32] Marlin, B. M. (2007). Collaborative filtering with multiple sparse observations. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI)*, 344–352.

[33] Meyn, S., & Tweedie, R. (2009). *Markov Chains and Stochastic Stability* (2nd ed.). Cambridge University Press.

[34] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.

[35] Norris, J. R. (1998). *Markov Chains.* Cambridge University Press.

[36] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362.

[37] McKinney, W. (2010). Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).

[38] Polymarket. (2024). *Polymarket Prediction Market Platform.* Retrieved from `https://polymarket.com`

[39] Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

[40] Ross, S. M. (2014). *Introduction to Probability Models* (11th ed.). Academic Press.

[41] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[42] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & van der Walt, S. J. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272.

[43] Scott, S. L. (2002). Bayesian methods for Hidden Markov Models: Recursive computing in the 21st century. *Journal of the American Statistical Association, 97*(457), 337–351.

[44] Sharan, R., & Karp, R. M. (2008). Modeling and analysis of biological networks. *PLOS Computational Biology*, 4(8), e1000116.

[45] Tetlock, P. E. (2004). *Expert Political Judgment: How Good Is It? How Can We Know?* Princeton University Press.

[46] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).

[47] Wolfers, J., & Zitzewitz, E. (2004). Prediction markets. *Journal of Economic Perspectives*, 18(2), 107–126.

[48] Wolfers, J., & Zitzewitz, E. (2006). Interpreting prediction market prices as probabilities. *NBER Working Paper No. 12200.*