Predicting Basketball Game Outcomes Using Markov Modeling

Fardin Haque
University of California San Diego
f1haque@ucsd.edu

Albert Chen
University of California San Diego
alc123@ucsd.edu

Mohid Tanveer
University of California San Diego
mtanveer@ucsd.edu

Aleksa Popovic
University of California San Diego
alpopovic@ucsd.edu

1 Problem Description

Basketball game outcomes are influenced not only by a team's demonstrated skill, but also game-to-game fluctuations, what fans often describe as "momentum." Our project investigates how these outcome patterns can be captured and predicted using probabilistic sequential modeling. Specifically, we study how a team's per-game performance, summarized through a statistic called the "Four Factors" (weighted average of a team's field goal percentage, turnover percentage, offensive rebounding, and free throws for each game), provides information about the underlying win/loss state of each game and how that latent state evolves over the course of a season [1] [2].

The importance of this problem lies in the real-world representation of competitive environments, which exhibit temporal dependencies where one game's outcome may affect the probability of winning the next. Performance metrics like "Four Factors" (denoted 4F) offer only a noisy reflection of a team's underlying competitive condition. Such models, which treat games as independent events, fail to capture these temporal dependencies or the uncertainty inherent in interpreting performance statistics.

By framing this task as a sequential prediction problem, we investigate the use of Markov Chains (denoted MC) and Hidden Markov Models (denoted HMM) [4]. MCs allow us to study how game outcomes transition over time, while HMMs naturally represent settings where hidden states (game outcomes) generate noisy observations (4F values). This provides a principled framework for reasoning about uncertainty, temporal structure, and the relationship between observed performance and unobserved competitive state. Our goal is to evaluate how effectively these probabilistic models capture momentum-like dynamics in basketball seasons and how accurately they can predict game sequences.

2 Data Sourcing and Processing

Our dataset is constructed from publicly available NBA box-score game logs on Basketball-Reference [3]. Using a web-scraping pipeline, we aggregate team-level regular-season games into a single CSV that, for each game, records the date, team, win/loss indicator, and the 4F statistics (eFG%, TOV%, ORB%, $\frac{FT}{FGA}$) [1]. The dataset contains 59,854 games spanning 25 seasons (2000–2024) and 37 unique teams, which we group into 686 team-season sequences for model training and evaluation.

For HMM-related pre-processing, we group games by season and compute per-season means and standard deviations of the 4F values across all regular-season games in that year. Each game's raw factor vector is converted to per-season z-scores, which control for long-term changes in league scoring environment and put the four features on a comparable scale. Since higher turnover percentage is worse, we flip the sign of the TOV% z-score so that larger values consistently indicate better offensive performance across all components.

We combine these standardized factors into a single scalar 4F performance score via a weighted linear combination, where the weights are learned by logistic regression on the training seasons (see Section 3.3.3). We then pool scores across seasons, compute empirical quantiles, and discretize the continuous values into K approximately equiprobable bins, yielding a discrete observation alphabet $\{0, 1, \ldots, K-1\}$. In all models, we partition seasons into training and evaluation splits. Seasons 2000-2017 and 2019–2023 are used for training, while seasons 2018 and 2024 are held out as test seasons shared across all models.

3 Methodology

3.1 Configuration

3.1.1 MCs

For our unigram MC model, each state S_i will be the outcome of the game for a given team, either a win or a loss. The initial probability distribution, which in the case of a unigram model is a stationary distribution, is the probability that a given game ends in a win or a loss.

For our bigram MC model, each state S_i will be the outcome of the game for a given team. The initial probability distribution is the probability that the first game of the season for a team will result in a win or a loss. The transition matrix will represent the probability that the team will win or lose game i given the outcome of game $i-1$.

For our trigram MC model, each state S_i will be the outcome of the game for a given team. The initial probability distribution is the probability that the first game of the season for a team will result in a win or a loss, and the probability that the second game of the season for a team will result in a win or a loss given the outcome of the first game. The transition matrix will represent the probability that the team will win or lose game i given the outcome of game $i-1$ and game $i-2$.

The CPTs for all three MC models can be found in Appendix Section E.

3.1.2 Threshold

In order to assess the effectiveness of using 4F scores in isolation, we construct a threshold model that predicts the outcome of a game using only the 4F score for that game. For our threshold model, we collect the 4F values per game (eFG%, TOV%, ORB%, FT/FGA). We then use these values to compute a single 4F scalar using the weights derived by Dean Oliver:

Four Factor	eFG%	TOV%	ORB%	FT/FGA
Weight	0.40	0.25	0.20	0.15

An optimal threshold is learned for each "tier" of teams such that if the team has a composite 4F score greater than that threshold, then a win is predicted; otherwise, a loss.

3.1.3 HMM

For our HMM, we assume a Markov structure $P(S_t \mid S_{t-1})$ with hidden states S_t and S_{t-1} and conditionally independent emissions $P(O_t \mid S_t)$ with observation O_t at game t.

Observations. For each season, we collect the 4F values per game, standardize each of the components within the season using z-scores, and then form a single composite 4F scalar using learned weights (see Section 3.3.3):

Four Factor	eFG%	TOV%	ORB%	FT/FGA
Weight	~ 0.50	~ 0.17	~ 0.16	~ 0.17

Concretely, letting z_{eFG}, z_{TOV}, z_{ORB}, and $z_{FT/FGA}$ represent the per-season z-scores. We compute:

$$z_{weighted} = 0.5z_{eFG} - 0.17z_{TOV} + 0.16z_{ORB} + 0.17z_{FT/FGA}$$

where the z_{TOV} is negated to ensure that larger values for all four components indicate stronger performance. We then pool these scores across seasons and apply quantile binning to obtain a discrete observation alphabet of size K yielding observation symbols in $\{0, 1, \ldots, K-1\}$ we run a grid search over values of K (e.g., 3, 5, ...) to select this parameter (see Section 4.3).

Hidden States. For the supervised setting, the hidden state is the binary game outcome per match (0 = loss, 1 = win). For the unsupervised setting, we treat the HMM as having $K_{\text{hidden}} > 2$ latent states; we also run grid search over values (e.g., 8, 16, ...) to select this parameter, and later map each hidden state to a win/loss label using the training data.

Sequences. We build season sequences from a CSV of game logs (using regular season games only). Each sequence pairs the state path (W/L) with the binned observation path (using the appropriate observation symbols $\{0, \ldots, K-1\}$).

3.2 Inference

3.2.1 MCs

We will perform inference using our ngram models by using the trained CPTs and sampling from them to generate the next game prediction.

Overview of the process:

1. First, we find which n-gram model to use according to the "tier" of the team. We read the win count of the team and determine which tier they belong to, using the models from that tier accordingly.

2. Each of these n-gram models has an initial CPT $P(\text{outcome}_1 = W/L)$ from which we sample to get the first game prediction.

3. Subsequent game predictions use the corresponding transition matrix; for example, the bigram model has a CPT for $P(\text{outcome}_t = W/L \mid \text{outcome}_{t-1} = W/L)$ to sample from.

4. We do this for all 82 games in a season and compare the final predicted sequence with the actual season sequence.

3.2.2 Threshold

We perform inference with our threshold model using the following process:

1. We determine which "tier" the team is in based on the true win count of the team for that season. This tier determines the optimal threshold that is used for win prediction, as this optimal threshold is learned based on data from other teams in this tier.

2. For each game in the season we compute the composite 4F scalar.

3. If the 4F score for the game is higher than the optimal threshold a win is predicted for that game. Otherwise a loss is predicted.

4. We perform this for all 82 games in the season, then compare the accuracy of the predictions with the true outcomes from the actual season.

3.2.3 HMM

For each test season observation sequence, we decode the most likely state path using Viterbi in log-space for numerical stability. Inputs are the learned B, π, A parameters from the learning step and an integer observation array. The algorithm returns the argmax state path

$$\arg\max_{S_{1:T}} \log \pi_{S_1} + \sum_{t=2}^{T} \log A_{S_{t-1},S_t} + \sum_{t=1}^{T} \log B_{S_t,O_t}.$$

In the supervised setting, these states are already binary W/L labels. In the unsupervised setting, the raw hidden-state path is first mapped to $\{0,1\}$ using a state-to-outcome mapping learned on the training data (see Section 3.3.3). We evaluate prediction quality by per-game accuracy against the true W/L states, aligned per sequence (element-wise match rate across all test sequences).

3.3 Learning

3.3.1 MCs

To train the CPTs of the unigram, bigram, and trigram models we use the counts of observed sequences of game outcomes for teams across several seasons. We first cluster teams into K "tiers" which represent their caliber by wins. Each cluster has its own n-gram trained for it, as seen in the figure for MC inference. We will use all the teams in a cluster for the counts of the n-gram. Examples of counts we take:

$$P(\text{outcome}_1 = w) = \frac{count(w)}{\text{seasons}}$$

$$P(\text{outcome}_t = w_1 | \text{outcome}_{t-1} = w_2) = \frac{count(w_1, w_2)}{count(w_2)}$$

So when we have K clusters, we have K different n-grams to choose from when doing inference. The results from partitioning the data into K clusters can be found in Appendix Section D.

3.3.2 Threshold

To learn the optimal threshold to be used by our threshold model, we use the K "tiers" of clusters as described in Section 3.3.1. For each team in each cluster, we go through every game of that season and collect their composite 4F score for that game and the resulting outcome (win or loss). We then construct a set of thresholds to consider by getting all unique 4F scores for that cluster, sorting them, then taking all midpoints between consecutive scores. We try each of these thresholds to predict the outcomes of all games in the cluster and keep the threshold that results in the best accuracy in predicting the outcomes. This optimal threshold becomes the threshold that is used at inference time for the corresponding tier a team is in.

3.3.3 HMM

Four-factor weighting via logistic regression. Before estimating HMM parameters, we learn the 4F combination weights using logistic regression. For each training-season game, we take the per-season z-scored 4F vector and fit a logistic regression model

$$P(outcome = 1 \mid z) = \sigma(w^\top z),$$

where *outcome* is the win indicator and σ is the sigmoid function. We train this model on training seasons, and then normalize the learned coefficients, which are used to construct the scalar 4F feature used by the HMM.

Supervised HMM learning. We then train the HMM in a supervised fashion, where the model behaves as a fully-observed MC on the training set: we observe both the W/L state and the discretized 4F symbol at each time step and estimate parameters via Maximum Likelihood Estimation with Laplace smoothing. During supervised training, we have 2 "hidden states" that map directly to W/L, and K observation symbols (or bins for 4F values). The **initial distribution** π is given by the relative frequencies of the first game's state across all training sequences, with smoothing. The **transition matrix** A is obtained from counts of adjacent state pairs $(S_t \to S_{t+1})$ aggregated across all training sequences, with smoothing and row normalization. The **emission matrix** B is computed by, for each state s, counting how often each observation symbol o occurs when $S_t = s$, with smoothing and row normalization. Formally, with smoothing $\alpha = 1$:

$$\hat{\pi}_s = \frac{\text{count}(S_1 = s) + \alpha}{\sum_{s'} \text{count}(S_1 = s') + 2\alpha},$$

$$\hat{A}_{ij} = \frac{\text{count}(S_t = i, S_{t+1} = j) + \alpha}{\sum_{j'} \text{count}(S_t = i, S_{t+1} = j') + 2\alpha},$$

$$\hat{B}_{s,o} = \frac{\text{count}(S_t = s, O_t = o) + \alpha}{\sum_{o'} \text{count}(S_t = s, O_t = o') + K\alpha}.$$

Unsupervised HMM learning (Baum–Welch). For the unsupervised models, we train HMMs on the observation sequences alone using the Baum–Welch algorithm (EM). For a given choice of hidden state count $N_{\text{states}} > 2$ and observation alphabet size K, we initialize π, A, and B as random stochastic matrices and run a scaled forward–backward algorithm to compute, for each sequence, the posterior probabilities

$P(S_t = i \mid O_{1:T})$ and $P(S_t = i, S_{t+1} = j \mid O_{1:T})$. We then update the parameters by aggregating these posteriors across all sequences:

$$\hat{\pi}_i = \frac{\sum_{\text{seq}} P(S_1 = i \mid O_{1:T}^{(\text{seq})})}{\sum_{i'} \sum_{\text{seq}} P(S_1 = i' \mid O_{1:T}^{(\text{seq})})},$$

$$\hat{A}_{ij} = \frac{\sum_{\text{seq}} \sum_{t=1}^{T^{(\text{seq})}-1} P(S_t = i, S_{t+1} = j \mid O_{1:T}^{(\text{seq})})}{\sum_{j'} \sum_{\text{seq}} \sum_{t=1}^{T^{(\text{seq})}-1} P(S_t = i, S_{t+1} = j' \mid O_{1:T}^{(\text{seq})})},$$

$$\hat{B}_{i,o} = \frac{\sum_{\text{seq}} \sum_{t:O_t^{(\text{seq})}=o} P(S_t = i \mid O_{1:T}^{(\text{seq})})}{\sum_{o'} \sum_{\text{seq}} \sum_{t:O_t^{(\text{seq})}=o'} P(S_t = i \mid O_{1:T}^{(\text{seq})})},$$

with a small smoothing constant added to avoid zero probabilities. We iterate these EM steps until the total log-likelihood of the training sequences converges. Because the hidden states in this setting are unlabeled, we finally learn a post-hoc mapping from hidden states to $\{0, 1\}$ by running Viterbi on the training observations, counting co-occurrences of hidden states and true W/L labels, and assigning each hidden state to its majority outcome. This mapping is then applied to Viterbi paths on the test set, yielding predicted W/L sequences that are evaluated with the same per-game accuracy metric as in the supervised case.

4 Results and Discussion

An accuracy metric is computed to represent how well each model does in predicting a sequence of games compared to the actual season record. As mentioned we are using team season data from 2018 and 2024 to test our models. One season is computed as follows:

1. We generate a predicted sequence of 82 games using our models, example output:
   Team 1 predicted season 2018: [0, 1, 1, 0, 1, 0, 0, 0, . . .] (total of 82 entries)

2. We compare this to the actual record of the team, for example: Team 1 actual season 2018: [1, 0, 1, 1, 0, 0, 0, . . .] (total of 82 entries)

3. We compute the accuracy metric as

   $$accuracy_{team\ 1,2018} = \frac{\text{correct predictions}}{82}$$

   which yields a value between 0.0 and 1.0, where 1.0 represents all 82 games were predicted correctly, and in order.

4. We compute this metric for all 30 teams in a season, calculating the average across all teams in a season to represent the model's performance for that season.

   $$performance_{2018} = \frac{\sum_{i=1}^{30} accuracy_{team\ i,2018}}{30}$$

4.1 MCs

In order to find the average performance across the different MC models, we tune the different parameters of the model and use the best configuration for our final average accuracy reporting. The different parameters of the MC models include:

- Ngram degree: unigram, bigram, trigram
- Number of clusters used for training: 2, 3, 4, 5

Parameter Analysis: We train unigram, bigram, and trigram models using MLE, each with different cluster numbers (find the exact results in the GitHub repo).

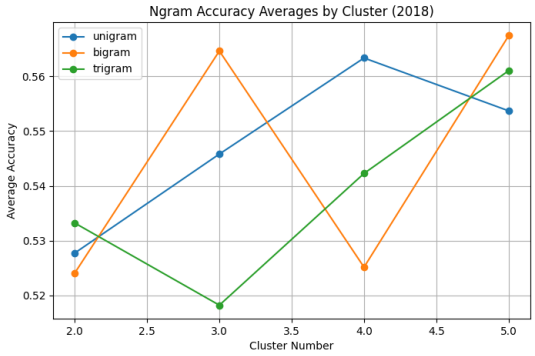

Figure 1: Accuracy of n-gram results on 2018 season when configured with different numbers of clusters in training.

When comparing each n-gram model when trained with different cluster numbers, we see that there is no clear correlation between average accuracy and cluster number. Results for 2018 and 2024 seem to hover between 0.5 and 0.6 for all cluster sizes and models (see Figure 1 for 2018 results, Appendix Section C for 2024 results). However, there is a slight increase as clusters become larger.

When taking a closer look at how accuracy is impacted within a cluster, we see why the average accuracy across clusters does not vary significantly. For brevity, we omit n-gram accuracy by tier for 2024 data in the main text, as it is similar to 2018 (see Appendix Section A for 2024 results). When looking at each model and its performance across the different tiers of each cluster, we see the same trend across all models: the accuracy starts off high for low tier teams, dips for mid tier teams, and rises again for high tier teams seen in Figure 2. Unigram and trigram accuracy plots, which exhibit similar behavior, are deferred to Appendix Section B. This is most likely because there are the most teams in mid tier clusters as seen in Appendix Section D. This could dilute CPTs and make predictions difficult to make accurate, whereas fewer teams in high and low tier partitions make trends in CPTs more apparent. This causes the average to be similar, as results are approximately symmetric.

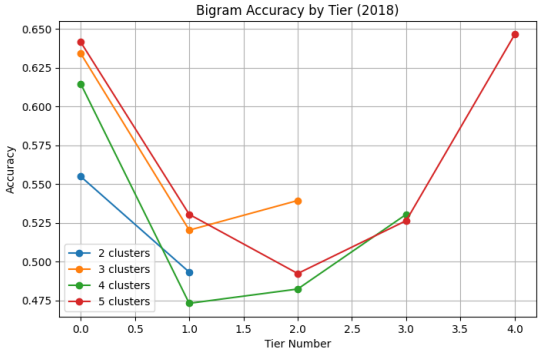

Figure 2: Bigram accuracy using differently clustered models across different team tiers.

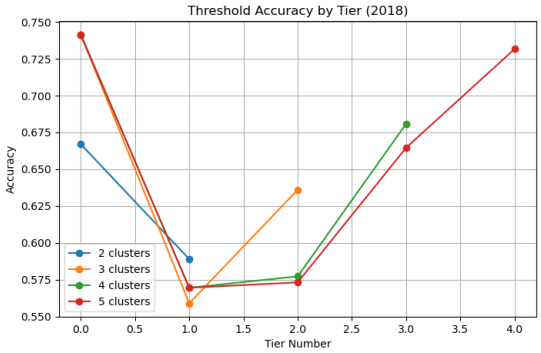

Figure 3: Threshold accuracy on the 2018 season using differently clustered models across different team tiers.

Based on parameter analysis, we use a bigram model with 5 clusters for our final results, which can be seen in Table 1.

Table 1: Final Results for MCs (5 clusters)

Season:	2018	2024	Both
Average Accuracy (0-1):	0.567	0.562	0.565
Mean of games predicted correctly (0-82):	~ 46	~ 46	~ 46

4.2 Threshold

We train the threshold model on different cluster numbers to see how performance differs. See Appendix Section G for the results.

When comparing the threshold results across different numbers of clusters, we see that similar to the n-gram MC models there is no clear correlation between average accuracy and cluster number. In fact, the accuracy behavior mimics the trend of the bigram MC model. However, the average accuracy of the threshold model was able to outperform that of the n-n-gram MC models at all cluster numbers, hovering between 0.6 and 0.7.

In addition, we examine how the accuracy of the threshold model differs between each "tier" of teams.

When looking at the performance in Figure 3, we again see similar trends as the n-gram MC models in that the performance starts off strong for low-tier teams, drops for the mid-tier teams, and then rises for high-tier teams. In this case, the accuracy of the threshold model was able to outperform that of the ngram MC models at all tiers. We predict that this behavior is due to the distribution of wins and losses for varying tiers. For low-tier teams, there is a high distribution of losses, and for high-tier teams there is a high distribution of wins. These extreme distributions make it easier to learn a threshold that can achieve high performance for these tiers, as wins and losses are more separable based on the 4F scores. Low-tier teams are unlikely to win games with a low 4F score, and similarly high-tier teams are unlikely to lose games with a high 4F score. For mid-tier teams, the distribution of wins and losses will be more evenly distributed, and thus it is difficult to learn a good threshold. Mid-tier teams are more likely to win with a low 4F score and lose with a high 4F score, and so wins and losses are not as easily separable by the 4F score. (Note that the results from the 2024 season are omitted from this section due to their similarity to the 2018 results; see Appendix Section F for 2024 results.)

These results, when compared to the n-gram MC model results, provide evidence that in isolation, using the 4F score observations to predict the outcome of a game may serve as a better model than using the results of the n previous games.

Based on parameter analysis we use a threshold model with 5 clusters for our final results, which can be seen in Table 2.

Table 2: Final Results for Threshold Model (5 clusters)

Season:	2018	2024	Both
Average Accuracy (0-1):	0.655	0.661	0.658
Mean of games predicted correctly (0-82):	~ 54	~ 54	~ 54

4.3 HMM

For our HMM configurations we first performed a grid search over the number of hidden states and observation bins in order to understand how model capacity and discretization affect predictive performance. Specifically, we varied:

- N_{states}: $\{2, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88\}$
- K (bin count): $\{3, 5, 8, 9\}$.

We compare configurations using average per-game accuracy across the two test seasons (2018 and 2024). We restrict to even hidden-state counts so that the post-hoc mapping from hidden states to $\{0, 1\}$ does not introduce an unpaired "extra" state that would collapse to a majority label and effectively

behave like a noisy coin-flip (empirically driving accuracy toward 0.50). As shown in Figure 4, the supervised HMM consistently dominates the unsupervised variants for a fixed bin count, and achieves its best performance with $K = 8$ bins (test accuracy ≈ 0.741). Supervised models with fewer bins (e.g., $K = 3$) lose a small amount of accuracy (down to ≈ 0.722), suggesting that overly coarse discretization discards useful structure in the 4F scores. For unsupervised HMMs, we observe that very small hidden-state counts underfit, whereas moderate-to-large state counts (e.g., $N_{\text{states}} \in [32, 88]$) manage to come closest to the supervised baseline at certain configurations, but still trail in the coarse grid search with our values. We also ran tests with higher state values and bins added in; these values, however, did not produce any significant results.

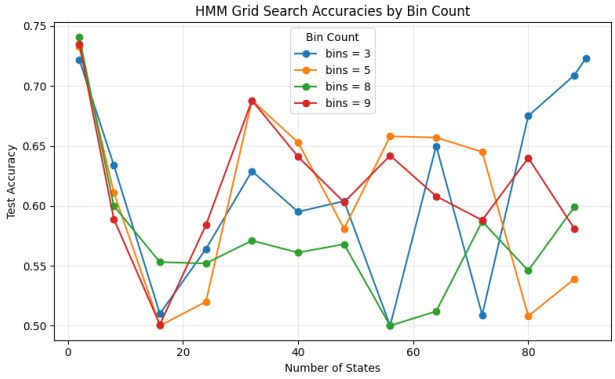

Figure 4: Accuracies for HMM configurations in grid search.

Motivated by these trends, we select a supervised configuration with $N_{\text{states}} = 2$ and $K = 8$ bins, and an unsupervised configuration with $N_{\text{states}} = 90$ and $K = 3$ bins (the latter from a finer sweep around the most promising unsupervised region). The final quantitative results are summarized in Table 3 and Table 4.

Table 3: Final Results for Supervised HMM

Season:	2018	2024	Both
Average Accuracy (0-1):	0.738	0.744	0.741
Mean of games predicted correctly (0-82):	~ 61	~ 61	~ 61

Table 4: Final Results for Unsupervised HMM

Season:	2018	2024	Both
Average Accuracy (0-1):	0.716	0.731	0.723
Mean of games predicted correctly (0-82):	~ 59	~ 60	~ 59

Comparing the results between the two configurations, we see that the accuracies are relatively close: only 0.018 points separate the total averages, corresponding to the supervised model predicting roughly 2 additional games correctly out of an 82-game season. Both HMM variants substantially outperform the n-gram MC baseline (Table 1) and the threshold baseline (Table 2), highlighting the value of jointly modeling temporal dependence and 4F observations. At a per-sequence level, the best seasons under both supervised and unsupervised training achieve accuracies in the 0.83–0.85 range, whereas the worst seasons hover just above 0.52–0.56, typically corresponding to more volatile, mid-tier teams whose alternating win/loss patterns are inherently harder to predict (see Appendix Section H).

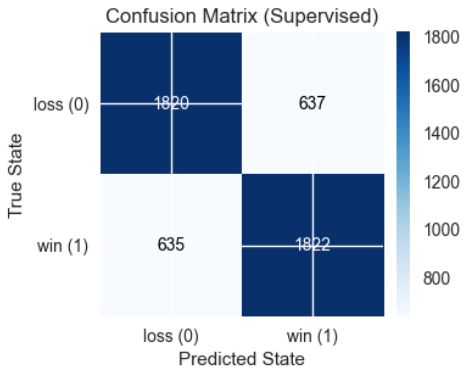

Figure 5: Confusion matrix for supervised HMM.

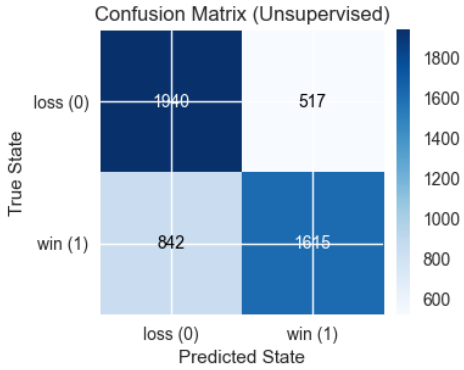

Figure 6: Confusion matrix for unsupervised HMM.

Looking at the confusion matrices in Figures 5 and 6, both models place most of their mass on the diagonal, indicating that they are not simply learning a trivial always-win or always-loss strategy. The supervised HMM reduces off-diagonal errors slightly more than the unsupervised HMM, consistent with its higher overall accuracy. Qualitatively, the learned transition and emission behavior captures intuitive "momentum" patterns: for high-performing teams, sequences often remain in the win state for extended stretches with high-probability self-transitions, while low-performing teams exhibit the opposite behavior; mid-tier teams show more frequent switching, where even a strong 4F game can be followed by a loss and vice versa. Sequence visualizations in Appendix Section I and J illustrate that the supervised and

unsupervised models track many streaks correctly, but can also lag or overreact around sharp changes in form.

From an optimization perspective, supervised HMM learning reduces to normalized counting with Laplace smoothing and is therefore extremely stable and inexpensive to run. Unsupervised Baum–Welch training is more computationally demanding, scaling roughly as $O(N_{\text{states}}^2 KTI)$ where T is the total number of games and I the number of EM iterations, but remained tractable for our dataset ($\approx 60{,}000$ games) and state counts up to 90. In practice we did not encounter numerical instability due to our use of log-space Viterbi decoding, scaling in the forward–backward algorithm, and small additive smoothing. With substantially more data we expect supervised performance to remain robust and unsupervised performance to improve as latent states are better constrained by longer and more varied sequences, at the cost of a linearly increasing runtime for each EM iteration.

5 Conclusion

We find that the HMM outperforms both n-gram and threshold baselines. This is expected as the HMM incorporates both past game information and data from the current game when making predictions. The n-gram MCs base their predictions solely on the outcomes of previous games, essentially ignoring any information from the current game. The threshold baseline considers only the 4F score for a team's performance. This baseline shows promising results as this is a strong statistic that is known to explain a large part of winning in basketball. The HMM, however, uses both the observed 4F score of the current game and the outcome of the previous game, allowing it to weight information on multiple facets when making a prediction. One limitation of our approach is not considering other factors. There are other factors to winning a basketball game like which team is at home or away, the players on a team, etc. Overall, our results from the HMM are on par with or slightly above other models which report accuracies between 65% and 72% when predicting on the same problem [5] [6].

6 Reflections and Contributions

- **Suggestions for future students**: For others looking to work on a similar problem we believe the field of making predictions in sports is vast and allows one to make many insightful findings. Many pros of this topic include working with large datasets, using existing statistics that others have made, and finding non-trivial relationships in data. Naturally, working with such large datasets presents some technical challenges related to processing this information, ensuring many factors are taken into consideration, and debugging problems when they emerge.

- **Fardin**: Contributed a significant portion of the work for conducting experiments on n-gram MCs. Wrote the code for training, performing inference, and parameter finetuning on n-gram models. Wrote portions in the report discussing results and creating graphs/diagrams on configuration, learning, inference. Overall, I felt I was able to learn and experiment with how probabilistic learning performs on real data. It was an insightful experience to see how even simple models that count frequencies, when tuned correctly, can make fairly accurate predictions.

- **Albert**: Wrote the code for unigram and trigram models to be used in experiments. Wrote code for conducting baseline experiments with a 4F threshold model. Wrote portions in the report discussing the methodology of the ngram MC baselines used, the methodology of the threshold model and the results of the threshold model. Personally, I enjoyed the opportunity to apply what we had learned in class regarding HMMs and ngram MC models to a real-world project, especially regarding a topic of which I have great interest in. I was able to gain experience with constructing the necessary probability distributions from a raw dataset in order to use these models.

- **Aleksa**: Implemented the Viterbi decoding algorithm for the HMM, built the logistic-regression pipeline for optimizing 4F weights, and wrote the web-scraping pipeline used to collect and preprocess multi-season NBA game data. The project helped me really understand how sequence models behave with real sports data. I also learned how much good data cleaning and feature design matter, since the quality of scraped game logs directly affected every model we built.

- **Mohid**: Developed HMM training code using both MLE (supervised) and Baum–Welch (unsupervised), including mapping hidden states to outputs in the latter. Built data-preprocessing pipelines with standardization and binning of 4F values, and created notebooks for training, evaluating, and tuning HMMs with visualizations and grid search. Contributed report sections on the problem description, data preparation, and HMM methodology/results. I felt that the most insightful part of working on this project was navigating the nuances between supervised and unsupervised HMM training, especially when interpreting hidden states learned through Baum–Welch. Building the preprocessing and training pipelines also reinforced how sensitive sequential models can be to feature design and parameter tuning.

References

[1] Basketball Reference. Basketball stats glossary. https://www.basketball-reference.com/about/glossary.html.

[2] Basketball Reference. Four factors. https://www.basketball-reference.com/about/factors.html. Accessed: 2025-02-06.

[3] Basketball Reference. Nba league index. https://www.basketball-reference.com/leagues/. Accessed: 2025-02-06.

[4] Geeks for Geeks. Hidden markov model in machine learning. *https://www.geeksforgeeks.org/machine-learning/hidden-markov-model-in-machine-learning/*, 2025.

[5] Matthew Houde. Predicting the outcome of nba games. *https://digitalcommons.bryant.edu/cgi/viewcontent.cgi?article=1000&context=honors_data_science*, 2021.

[6] Teno González Dos Santos, Chunyan Wang, Niklas Carlsson, and Patrick Lambrix. Predicting season outcomes for the nba. *https://liu.diva-portal.org/smash/get/diva2:1656665/FULLTEXT01.pdf*, 2022.

A Average Accuracy across Clusters for MCs (2024)

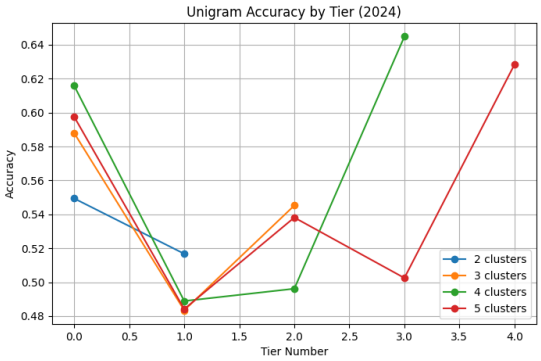

Figure 7: Unigram accuracy on 2024 season using differently clustered models across different team tiers.

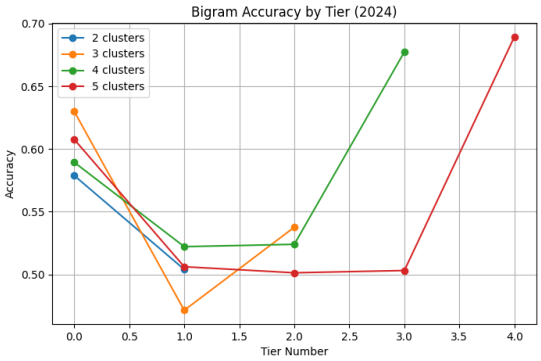

Figure 8: Bigram accuracy on 2024 season using differently clustered models across different team tiers.

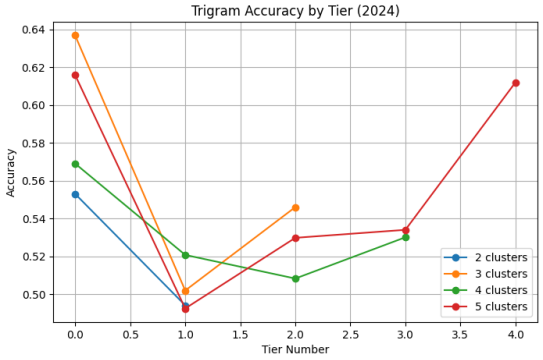

Figure 9: Trigram accuracy on 2024 season using differently clustered models across different team tiers.

B Average Accuracy across Clusters for MCs (2018)

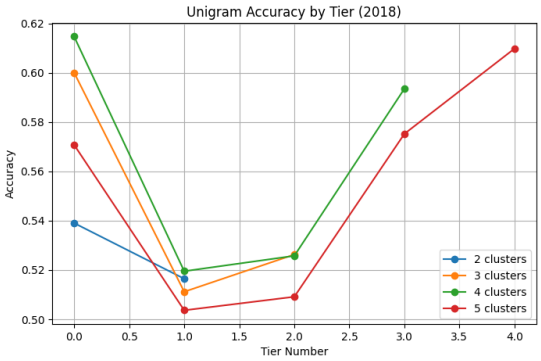

Figure 10: Unigram accuracy on 2018 season using differently clustered models across different team tiers.

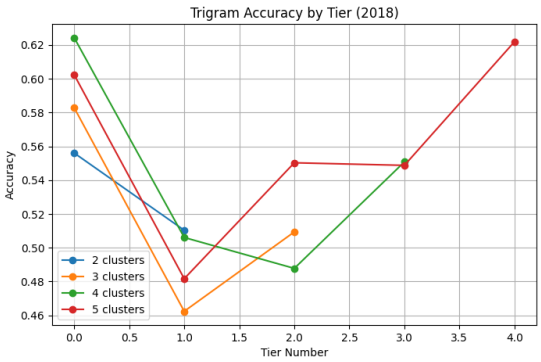

Figure 11: Trigram accuracy on 2018 season using differently clustered models across different team tiers.

C Average Accuracy by Tiers for MCs (2024)

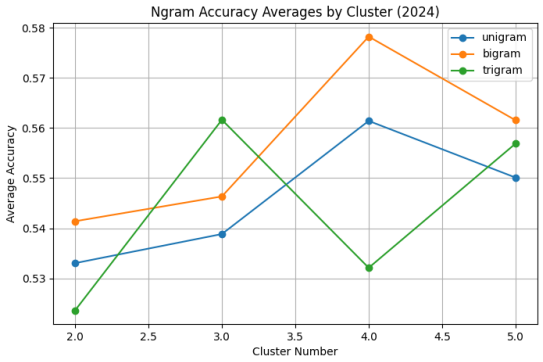

Figure 12: Accuracy of n-gram results on 2024 seasons when using the same models as Figure 1.

D Cluster Partitions

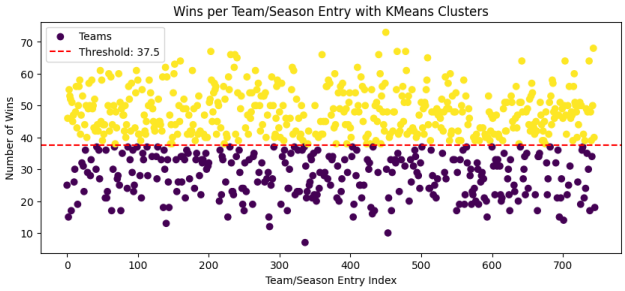

Figure 13: Clustering of teams by wins with 2 clusters.

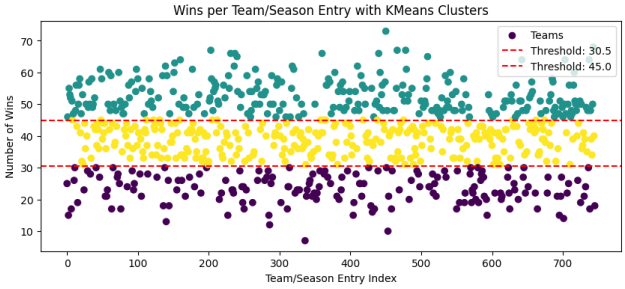

Figure 14: Clustering of teams by wins with 3 clusters.

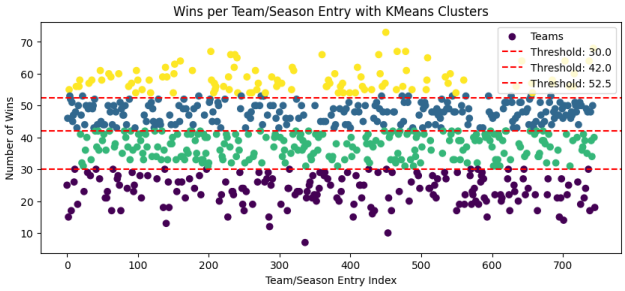

Figure 15: Clustering of teams by wins with 4 clusters.

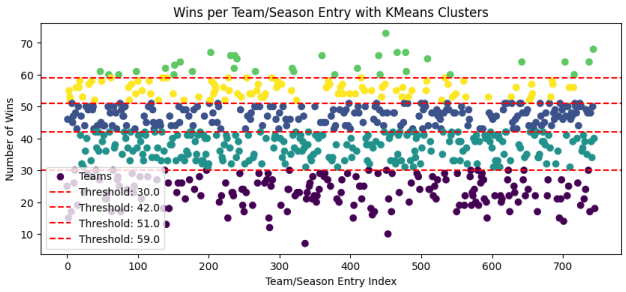

Figure 16: Clustering of teams by wins with 5 clusters.

E CPTs for N-gram Models

E.1 Unigram

Table 5: CPTs for the unigram MC model.

$P(S_i)$
W
L

E.2 Bigram

Table 6: CPTs for the bigram MC model.

$P(S_1)$	$P(S_i \mid S_{i-1})$	
W	W	W
L	L	W
	W	L
	L	L

E.3 Trigram

Table 7: CPT's for the trigram MC model.

$P(S_1)$	$P(S_2 \mid S_1)$		$P(S_i \mid S_{i-1}, S_{i-2})$		
W	W	W	W	W	W
L	L	W	L	W	W
	W	L	W	L	W
	L	L	L	L	W
			W	W	L
			L	W	L
			W	L	L
			L	L	L

F Average Accuracy by Tiers for Threshold Model (2024)

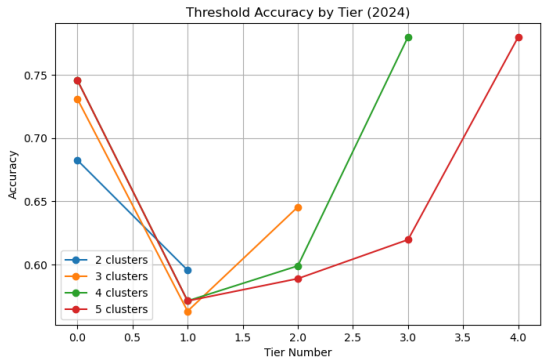

Figure 17: Threshold accuracy on the 2024 season using differently clustered models across different team tiers.

G Average Threshold Model Accuracy across Clusters

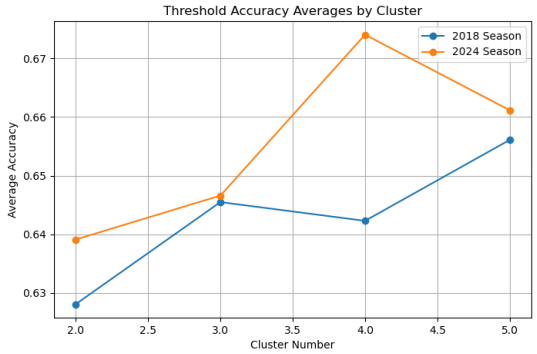

Figure 18: Average threshold accuracy on 2018 and 2024 seasons when configured with different numbers of clusters.

H Per Sequence Accuracy Statistics for HMMs

Table 8: Best and worst test sequences for supervised HMM.

Group	Rank	Season	Index	Accuracy
Worst sequences				
	1	2018	20	0.524
	2	2018	23	0.634
	3	2024	57	0.639
	4	2018	29	0.646
	5	2018	5	0.671
Best sequences				
	1	2018	14	0.854
	2	2024	59	0.841
	3	2024	51	0.841
	4	2018	2	0.841
	5	2024	39	0.841

Table 9: Best and worst test sequences for unsupervised HMM.

Group	Rank	Season	Index	Accuracy
Worst sequences				
	1	2018	7	0.549
	2	2018	20	0.561
	3	2024	38	0.593
	4	2018	23	0.622
	5	2024	36	0.646
Best sequences				
	1	2018	14	0.854
	2	2024	39	0.854
	3	2024	51	0.854
	4	2024	52	0.854
	5	2018	9	0.829

I Supervised HMM Sequence Prediction Visualization

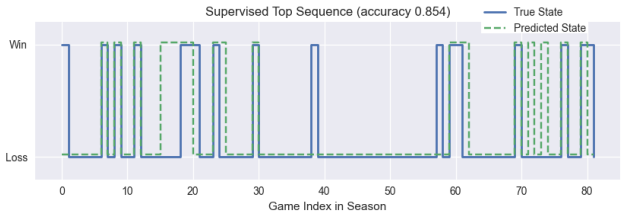

Figure 19: Top test sequence visualized for HMM trained with supervised learning.

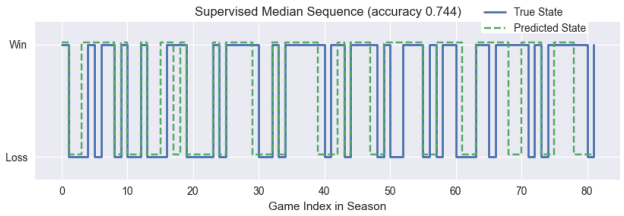

Figure 20: Median test sequence visualized for HMM trained with supervised learning.

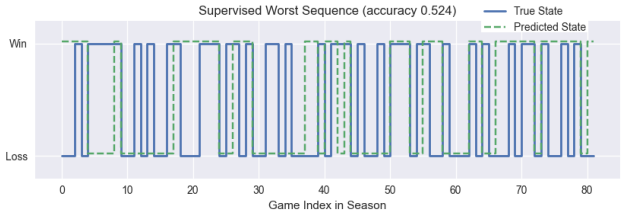

Figure 21: Worst test sequence visualized for HMM trained with supervised learning.

J Unsupervised HMM Sequence Prediction Visualization

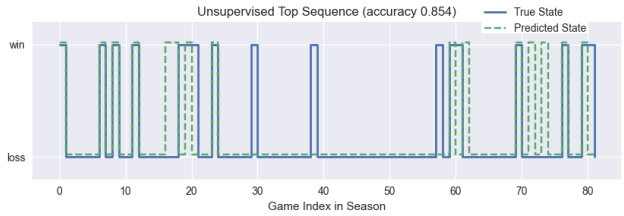

Figure 22: Top test sequence visualized for HMM trained with unsupervised learning.

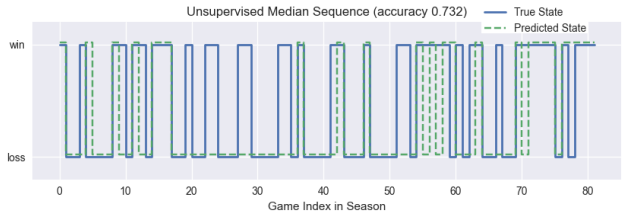

Figure 23: Median test sequence visualized for HMM trained with unsupervised learning.

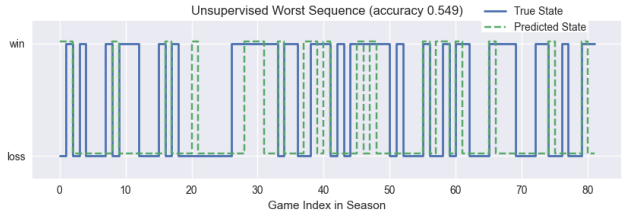

Figure 24: Worst test sequence visualized for HMM trained with unsupervised learning.