AML Final Project (CS5785/ORIE5750/ECE5414)

Alex Bean (atb95) Donovan McManus (dpm262) Kamron Vaswani (kv277)

In this project, we contributed equally across all areas. We all participated in brainstorming to drive methodologies and decipher which is best for our data-augmentation processes to develop a robust solution strategy and shared the responsibility of coding. There was no structured way of assigning things, just people picking up and building whenever their schedule permitted. For the write-up, we collectively crafted our insights that allowed us to leverage everyone's strengths and point of view to ensure a cohesive representation of our work.

Abstract

We developed a machine learning pipeline to predict NBA game scores relative to the betting over/under line. Preprocessing focused on creating a clean dataset by isolating games from the 2013 season onward, accounting for the NBA's evolving playstyle. Feature engineering included calculating 15-game averages for points scored, point differentials, and win percentages to capture recent performance trends. Categorical features like team and season identifiers were one-hot encoded, while binary encoding of player participation captured individual contributions. Initial experiments evaluated Lasso, ElasticNet, and Random Forest, both with and without one-hot encoding of player and season data. ElasticNet emerged as a strong contender with a Mean Squared Error (MSE) of 333.3, leading us to select it for further analysis. Using a dataset of 37,105 rows, we randomly sampled 1,000 rows for testing, leaving 36,105 for training. ElasticNet achieved an accuracy of 53.1%, exceeding the profitability threshold of 52.4% in sports betting. We implemented hyperparameter tuning for ElasticNet, Lasso, Random Forest, and XGBoost, optimizing parameters like `alpha` *and* L_1*-ratio for ElasticNet. The final evaluation highlighted K-Nearest Neighbors (KNN) as the top-performing model with an accuracy of 52.9%, closely followed by ElasticNet and Random Forest. These results were strong given our limited experience with this approach. We plan to continue refining our work beyond this final submission, with a particular focus on preprocessing our data to enhance our chosen model's accuracy.*

Methodologies

Pre-Processing Methods

The preprocessing pipeline for this project focused on transforming raw game and player data into a clean and feature-rich dataset. The first step involved filtering data to ensure consistency by isolating games from the 2013 season onwards. We chose 2013 because, given the NBA's evolution in recent years in terms of style of play and coaching strategies, particularly with the widespread adoption of three-point shooting, we wanted to ensure stability by avoiding biases introduced by older playing styles. With this, we restricted our dataset for computability reasons with the extensive data sets we were working with. Filtering was applied across multiple datasets — `game_summary`, `game_data`, and `players` — by converting date columns to datetime format.

Synchronizing this filtering step, we dove into feature engineering. Specifically, we calculated the average points scored by teams over their last 10 games, which ensured fallback values were applied when teams had fewer than 15 games as well as over/under interpretability. Breaking down our data set into these 15 game increments allowed us to capture more relevant and recent performance trends while mitigating the noise from earlier seasons. This value of 15 was one that was chosen as it yielded the best Mean Squared Error (MSE) which gave us the best opportunity to predict future game outcomes more accurately.

Similarly, we introduced the 15-game point differential to capture the difference between points scored and points conceded and computed the win percentage for both home and away teams. Metrics for shooting performance such as three-point shots made (`fg3m_home` and `fg3m_away`) and field goals made (`fgm_home` and `fgm_away`) were also

calculated using row-level operations. These engineered features required iterating over game records and aggregating recent performances, which, while computationally intensive, were necessary for us to gather team trends.

To integrate player data into the game-level dataset, we merged the `players` dataframe with the team data, aligning on team IDs and filtering by season. We then applied a binary encoding approach that indicated player participation in specific games. More specifically, for each game row, relevant players were identified, and a `1` was assigned under their respective columns. Our focus on player statistics was an experiment to help understand how our models performed at different levels of granularity. This is something we will discuss further in our results section.

Handling missing values was another crucial step. After identifying that features such as `home_3_point_last_15` and `visitor_fg_last_15` had 11 missing entries, these rows were dropped because of how few there were. Categorical variables like `home_team_id`, `visitor_team_id`, and `season` were transformed using one-hot encoding, which expanded the dataset to 1,411 columns. This quelled any computability errors as machine learning algorithms are, as we know it, obviously better with numbers than words.

Our preprocessing pipeline produced a clean and structured dataset that integrated game summaries, player statistics, and recent performance trends. This balanced foundation allowed us to focus on developing the conceptual aspect of this project: determining whether an NBA game's score will be over or under the betting line.

One-Hot Encoding Analysis

Our first step in testing the validity of our preprocessing steps was to split our data into two training sets that were tested on three regression models: LASSO, ElasticNet, and Random Forest. Furthermore, these models were evaluated in two settings: one without player and season-specific features being one-hot encoded and another with player and season data one-hot encoded.

In the setting without player and season-specific encoding, Random Forest slightly outperformed the linear models, achieving a MSE of 337.5. In comparison, ElasticNet earned a MSE of 341.4 and LASSO earned a MSE of 341.8. When testing these models where player and season data were one-hot encoded, the MSE improved slightly for all models: LASSO reached 335.7, ElasticNet achieved 333.3, and Random Forest scored 338.9. This marginal improvement for the majority of our models (LASSO and ElasticNet), with Random Forest slightly worsening, suggested to us that we should lean into our one-hot encoded data set where player and season information enhanced prediction accuracy.

Feature coefficients from the linear models revealed key insights into the importance of specific players and teams. For example, certain player IDs and team performances were found to contribute significantly to the predictions. LASSO produced sparse models by reducing less significant feature coefficients to zero, while ElasticNet retained more features. Random Forest, as a non-linear model, was able to capture more complex relationships in the data but lacked the interpretability provided by the linear models. It was interesting to unpack these coefficients as they suggested that players like Terrence Jones or Danilo Gallinari strongly correlated to an increase in our target variable, total points scored. Using our knowledge of basketball, these specific player contributions were surprising findings, but something we noted nonetheless.

The inclusion of player participation, encoded as binary features, added predictive value. This not only underlined the importance of individual players on game outcomes, but also guided our analysis in picking levels of granularity for each step in how we wanted to best compute total score of a game. Complimenting this, encoding season-level information also contributed to reducing the MSE as we navigated the power of performance trends across seasons and games.

In revising our one-hot encoding process, we were moving forward knowing that Random Forest offered slightly better raw performance in the non-encoded setting while the linear models provided better, more interpretable results in our encoded setting.

Accuracy Check & Hypertuning

Following our one-hot encoding testing on LASSO, ElasticNet, and Random Forest, we decided to move forward with ElasticNet for training and evaluating predictive models for over/under sports betting predictions, as it had demonstrated the lowest MSE. The odds dataset used for this analysis contained 37,105 rows, representing a large pool of games and betting data. From this dataset, we sampled 1,000 random game IDs to create the test set, ensuring the remaining data (36,105 rows) was used for training. This split allowed us to evaluate ElasticNet on an unseen subset of the data. These predictions were added back to the test dataset and merged with the original odds dataset to create a direct comparison between predicted total points, actual points, and the over/under line. By focusing on a subset of 1,000 test samples, we could validate the model's generalizability without compromising computational efficiency (a large focus for us as we wrangled such large data sets and tested a variety of models).

To evaluate ElasticNet's performance in the over/under setting, we implemented a custom accuracy metric. This metric assessed whether the model's predictions correctly aligned with actual outcomes relative to the over/under line. In this scenario, the accuracy of the ElasticNet model was approximately 53.1%, which was strong according to exter-

nal literature. More specifically, as found in Andrew Josselyn's article (1), "How I Achieved a 56.4% ROI on Sports Betting," he states that "to be profitable [in] sports betting you have to have a win percentage (accuracy) of 52.4% ... This is impressive because only 3% of sports bettors are actually profitable." Without hypertuning, our accuracy was already above what is considered mathematically plausible in profitting off sports betting. This evidently indicated that we had strong predictive capability with what we had and, as a starting point, can expand on this for further optimization.

We expanded our analysis by introducing hyperparameter tuning to refine ElasticNet, Lasso, Ridge, RandomForest, GradientBoosting, SVR, KNN, MLP, and XGBoost in the context of over/under betting predictions. However, we began by utilizing our custom built function, `tune_single_model`, that is designed to test a single model with different hyperparameter configurations, evaluating its accuracy, MSE, and R^2. Although we initially standardized the input data using a `StandardScaler`, we did not yield as good of a MSE score, so we moved on without it. For each set of hyperparameters, the model is trained on the training data and used to predict outcomes on the test set. Predictions are then merged with the original odds data, allowing comparisons between actual and predicted values relative to the over/under line. A second function, `tune_models`, iterates through multiple models and their respective hyperparameter grids, systematically applying `tune_single_model` to each. The function also handles data splitting into training and testing subsets and ensures compatibility with different models and parameter configurations.

We continued by defining another function to plot the aforementioned models. In interpretting these results, we visualized the impact of hyperparameters on model performance. For ElasticNet, we focused on how variations in alpha and the L_1-ratio influenced accuracy. This was a wrestle match that helped us understand the trade-offs between sparsity and regularization. Furthermore, these visualizations highlighted the optimal balance of parameters for ElasticNet which helped in fine tuning our models. Similar visualizations were generated for other models, including Random Forest, Gradient Boosting, and Ridge regression. For Random Forest, we examined the relationship between the number of estimators, max depth, and accuracy. In Gradient Boosting models, parameters such as learning rate and max depth were explored to assess their influence on prediction accuracy. For Ridge and LASSO, alpha values were plotted against accuracy to understand how varying levels of regularization impacted performance.

Results & Discussion

Finally, it was time to dive into our results. Here, key trade-offs between simpler linear models like ElasticNet and Lasso and more complex non-linear models like Random Forest and SVR were revealed. However, before diving into these accuracy percentages, we must preface that our first tuned models were tuned on the data frame that did not include players and season as one-hot encoded. For our second set of tuned models utilizes the data frame that did one-hot encode players and season. We wanted to evaluate these differences again.

```
Tuning ElasticNet...
Best Params for ElasticNet: {'alpha': 0.24, 'l1_ratio': 1.0} with Accuracy: 0.518
Tuning Lasso...
Best Params for Lasso: {'alpha': 0.35} with Accuracy: 0.518
Tuning Ridge...
Best Params for Ridge: {'alpha': 100} with Accuracy: 0.506
Tuning RandomForest...
Best Params for RandomForest: {'n_estimators': 100, 'max_depth': None} with Accuracy: 0.519
Tuning GradientBoosting...
Best Params for GradientBoosting: {'n_estimators': 200, 'max_depth': 7} with Accuracy: 0.494
Tuning SVR...
Best Params for SVR: {'C': 0.1, 'epsilon': 1} with Accuracy: 0.474
Tuning KNN...
Best Params for KNN: {'n_neighbors': 7, 'weights': 'uniform'} with Accuracy: 0.529
Tuning MLP...
Best Params for MLP: {'hidden_layer_sizes': (150,)} with Accuracy: 0.504
Tuning XGBoost...
Best Params for XGBoost: {'n_estimators': 50, 'max_depth': 5, 'learning_rate': 0.1} with Accuracy: 0.501
```

Figure 1. Performance without one-hot encoded data.

For ElasticNet, the optimal parameters were `alpha=0.24` and L_1-ratio=1.0, achieving an accuracy of 0.518. Similarly, Lasso showed comparable performance with `alpha=0.35`, also achieving an accuracy of 0.518. Ridge regression performed slightly worse, with `alpha=100` resulting in an accuracy of 0.506. Random Forest performed well, with the best parameters being `n_estimators=100` and `max_depth=None`, yielding an accuracy of 0.519. Gradient Boosting achieved an accuracy of 0.494 with `n_estimators=200` and `max_depth=7`, slightly below Random Forest. Support Vector Regression (SVR) struggled compared to other models, obtaining an accuracy of 0.474 with `C=0.1` and `epsilon=1`. K-Nearest Neighbors (KNN) delivered the highest accuracy among all models at 0.529 with `n_neighbors=7` and uniform weights. The Multi-Layer Perceptron (MLP) neural network performed reasonably well, achieving an accuracy of 0.504 with hidden layer sizes of (150,). XGBoost showed moderate performance, reaching an accuracy of 0.501 with `n_estimators=50`, `max_depth=5`, and a learning rate of 0.1.

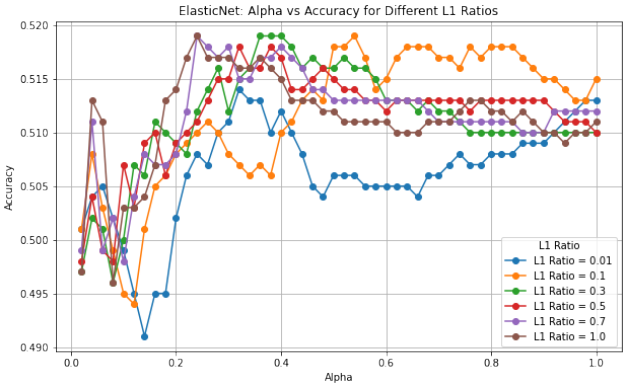
Flipping to the data set with players and season one-hot encoded, we got the following accuracy rates:

```
Tuning ElasticNet...
Best Params for ElasticNet: {'alpha': 0.24, 'l1_ratio': 0.7} with Accuracy: 0.519
Tuning Lasso...
Best Params for Lasso: {'alpha': 0.25} with Accuracy: 0.518
Tuning Ridge...
Best Params for Ridge: {'alpha': 1} with Accuracy: 0.509
Tuning RandomForest...
Best Params for RandomForest: {'n_estimators': 50, 'max_depth': 20} with Accuracy: 0.504
Tuning GradientBoosting...
Best Params for GradientBoosting: {'n_estimators': 50, 'max_depth': 5} with Accuracy: 0.512
Tuning SVR...
Best Params for SVR: {'C': 10, 'epsilon': 0.1} with Accuracy: 0.519
Tuning KNN...
Best Params for KNN: {'n_neighbors': 7, 'weights': 'uniform'} with Accuracy: 0.514
Tuning MLP...
Best Params for MLP: {'hidden_layer_sizes': (150,)} with Accuracy: 0.505
Tuning XGBoost...
Best Params for XGBoost: {'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.1} with Accuracy: 0.506
```

Figure 2. Performance with one-hot encoded data.

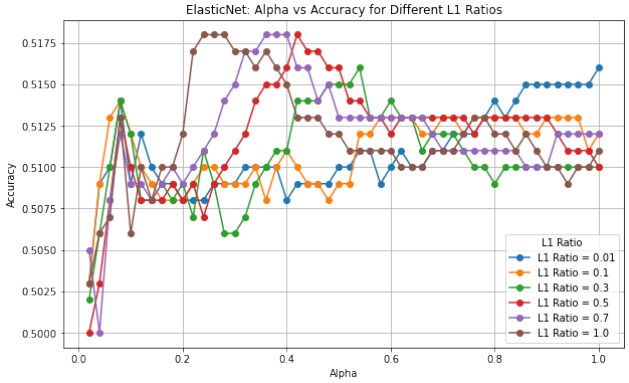
With this data frame, ElasticNet achieved an accuracy of 0.519 with optimal parameters `alpha=0.24` and L_1-ratio=0.7. Lasso followed closely with an accuracy of 0.518 using `alpha=0.25`, while Ridge regression showed slightly lower performance with an accuracy of 0.509 when `alpha=1`. Random Forest achieved an accuracy of 0.504 with `n_estimators=50` and `max_depth=20`, while Gradient Boosting performed slightly better with an accuracy of 0.512 when using `n_estimators=50` and `max_depth=5`. Support Vector Regression (SVR) demonstrated competitive performance, matching ElasticNet's accuracy of 0.519 with `C=10` and `epsilon=0.1`. K-Nearest Neighbors (KNN) also performed well, achieving an accuracy of 0.514 using `n_neighbors=7` and uniform weights. The Multi-Layer Perceptron (MLP) neural network achieved a moderate accuracy of 0.505 with hidden layer sizes of (150,). XGBoost demonstrated comparable performance with an accuracy of 0.506 when configured with `n_estimators=100`, `max_depth=3`, and a learning rate of 0.1.

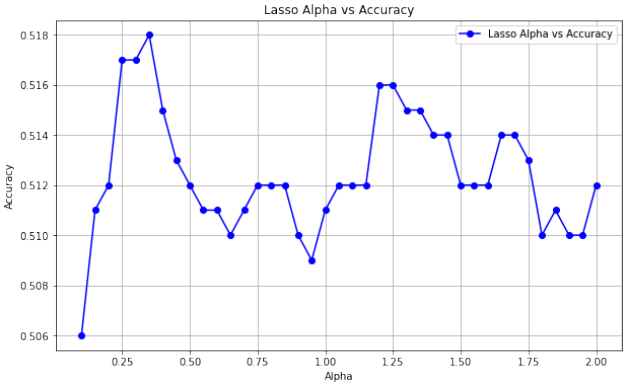
Based on these results, KNN emerges as the best model with an accuracy of 0.529. This performance suggests that KNN effectively captures the relationships within the data in this specific task. ElasticNet, Lasso, and Random Forest also performed well, highlighting their balance between complexity and generalizability. Ironically, our analysis underscores the potential of simpler models, such as KNN and ElasticNet regression, to compete effectively with more complex algorithms like Gradient Boosting and XGBoost.

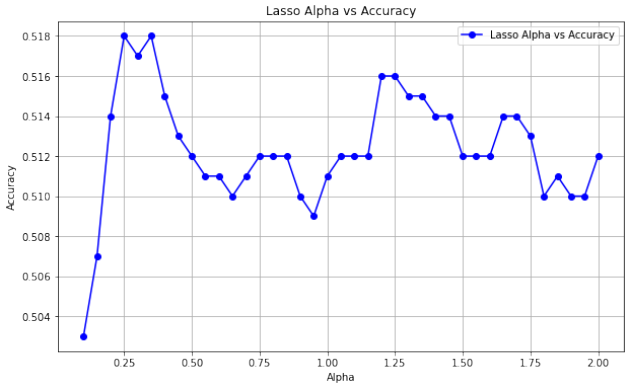


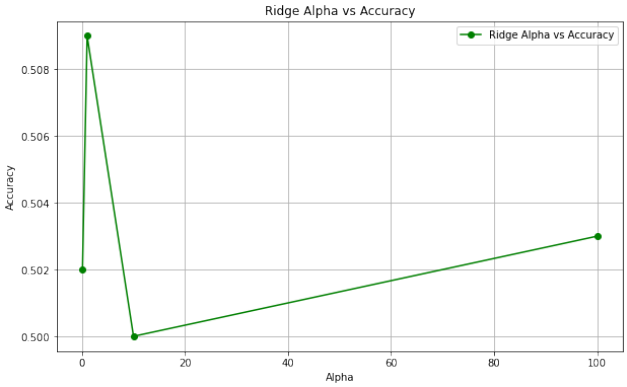
The first graph illustrates ElasticNet's performance across different α and L_1-ratio values without one-hot encoding player and season data, while the second graph presents results with one-hot encoding applied. Without encoding, the accuracy appears to plateau around 0.5175 for optimal α and L_1-ratio configurations, indicating decent performance but with noticeable noise, particularly at lower α values. This variability highlights the limitations of excluding detailed categorical features such as player and season data.

When one-hot encoding was applied in the second graph, the overall trends became smoother, and accuracy levels remained similar, peaking just below 0.52. This improvement in stability demonstrates how encoding player and season data added consistency to the model by capturing additional granularity and reducing noise. Interestingly, higher L_1-ratios performed better in both cases, likely due to their ability to enforce sparsity and prioritize more relevant features. The combined effect of encoding and optimized regularization contributed to more reliable performance, which was something that validated our decision to focus on one-hot encoded data.





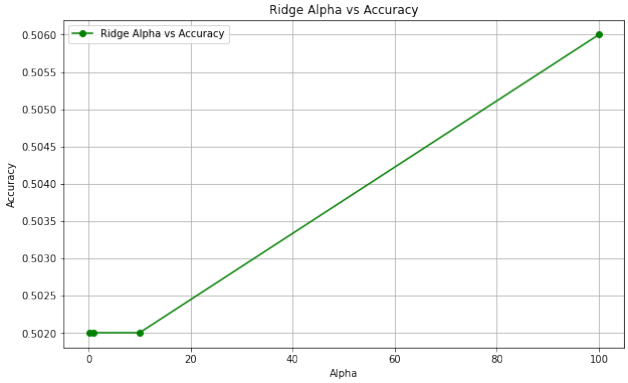




The first graph illustrates Lasso's performance across varying α values without one-hot encoding player and season data, while the second graph shows results with one-hot encoding applied. Without encoding, accuracy peaks at approximately 0.518 for an α around 0.25 but displays significant fluctuations as α increases. This inconsistency highlights the challenges in identifying stable patterns without the additional granularity provided by one-hot encoding.

In contrast, when one-hot encoding was applied, the second graph demonstrates a slighter higher peak in accuracy at 0.519 at $\alpha = 0.25$, but the overall trends are smoother. This suggests that encoding player and season data enhances the model's stability by capturing detailed categorical relationships. Despite these improvements in trend stability, the absolute accuracy gains are minimal, so the performance of Lasso is not dramatically influenced by player and season features. Together, both graphs reveal a decline in accuracy as α moves beyond 0.5. This suggests that higher regularization levels may overly constrain the model and reduce its predictive power.

Our first graph illustrates Ridge's performance across varying α values without one-hot encoding player and season data, while the second graph shows results with one-hot encoding applied. Without encoding, accuracy steadily increases from about 0.502 at $\alpha = 0$ to approximately 0.506 at $\alpha = 100$. This gradual improvement suggests that stronger regularization may stabilize predictions in the absence of granular categorical features. In contrast, when one-hot encoding is applied, the initial accuracy is higher (around 0.508 at $\alpha = 0$), but it rapidly declines for lower α values before slowly recovering to about 0.502 at $\alpha = 100$. This pattern indicates that while encoding player and season data can produce a higher starting accuracy, it introduces greater sensitivity to α and more volatility in the model's performance. Although accuracy eventually levels off, the net gain from encoding is minimal, and the model does not consistently surpass the non-encoded setting. Both graphs reveal that Ridge is relatively stable compared to other models, as changes in α yield only incremental shifts in accuracy. These results suggest that while encoding offers a slight edge in certain configurations, Ridge's performance does not significantly benefit from detailed categorical features and higher α values only modestly improve predictive accuracy.



References

[1] Josselyn, Andrew. ”How I Achieved a 56.4% ROI on Sports Betting.” *Medium*, 13 Sept. 2020, medium.com/@drewjosselyn/how-i-achieved-a-56-4-3b544e06d84b: :text=To%20be%20profitable%20sports%20betting,sports%20bettors%20are%20actually%20profitable.