XGB with Bayesian Optimization Summary

Table of Contents:

---

1. High-Level Summary

Goals:

1. Predict whether a favorited team will win an NFL game.
2. Why does this happen & what are the major components driving the outcome?

<u>High-Level Part 1:</u>

- Utilizing an XGBoost model with Bayesian Optimization, we can correctly predict 70.25% of game outcomes.

- Utilizing classification thresholds, we can predict 80% of the correct outcomes, but we are only able to implement the model on 48% of games.

<u>High-Level Part 2:</u>

***the pdp, shap & variable importance plots discussed are at the bottom of the section.

<u>LOGspread:</u>
The most important predictor in determining whether the favorite was going to win was LOGspread, which is a derivation of the Las Vegas Spread.

- A high LOGspread_favorite value increases the chances of the favorite winning.

- Once LOGspead reaches 1.5, the probability of the favorite winning increases from 0.20 to 0.45. As it reaches 2.0, the probability increases to 0.60.

Favorite Teams WR score:
The second most important indicator is the Favorite Teams WR score (fav_wr_score)

- This is derived from the Favorite WR madden rating minus the underdogs Secondary madden ratings. (In an game, the opposing secondary is tasked with covering the offenses WRs)

- A low fav_wr_score value decreases the chances of the favorite winning.

- As the score approaches 0 (even ratings), the probability of the favorite winning begins to increase.

Favorites average points scored while playing away:
The third most important indicator is the Favorites average points scored while playing away (fav_avg_points_scored_away).

- A high fav_avg_points_scored_away value increases the chances of the favorite winning.

- Once this feature reaches +21 the probability of the favorite winning spikes from 0.30 to 0.45.

- 21 average points scored away seems to be the number that drives the increase in the probability of the favorite winning

Underdog's average yards surrendered:
The fourth most important variable was the underdog's average yards surrendered.

- A high value increases the chances of the favorite winning.

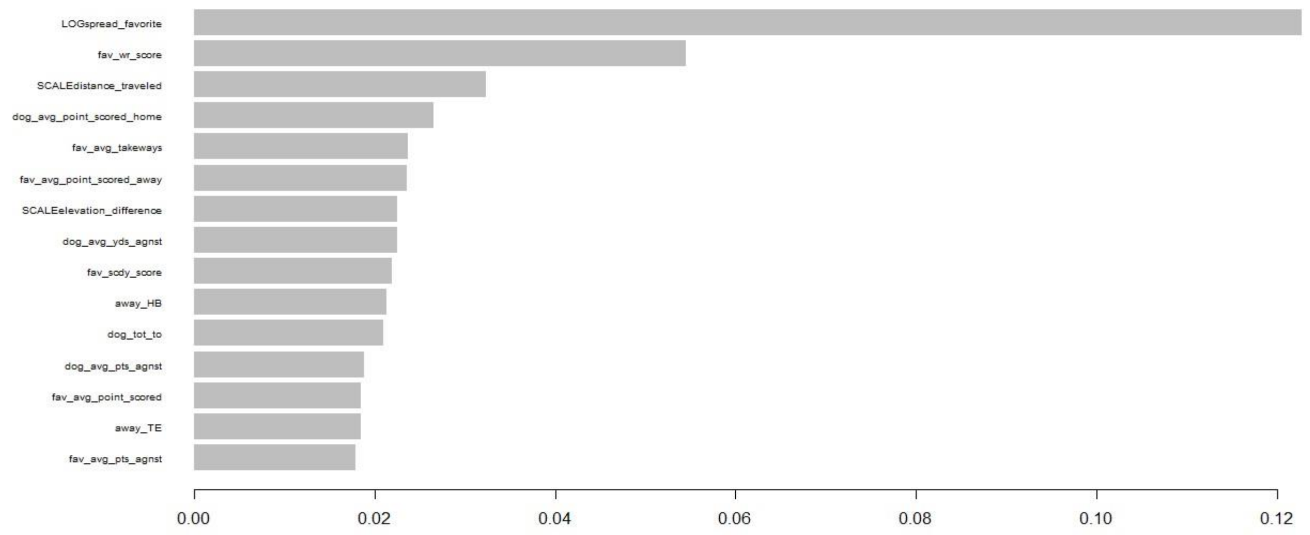
- Once this feature exceeds 375 yds, the probability of the favorite winning increases from 0.30 to 0.50.

Summary:
Items which signal the favorite is going to win
- LOGspread exceeds 1.5
- The favorite WR score is close to 0
- The favorite is averaging 21+ on the road
- On average, the underdog is giving up around 375 yards a game

visuals on next page*

Visuals:

Variable Importance:

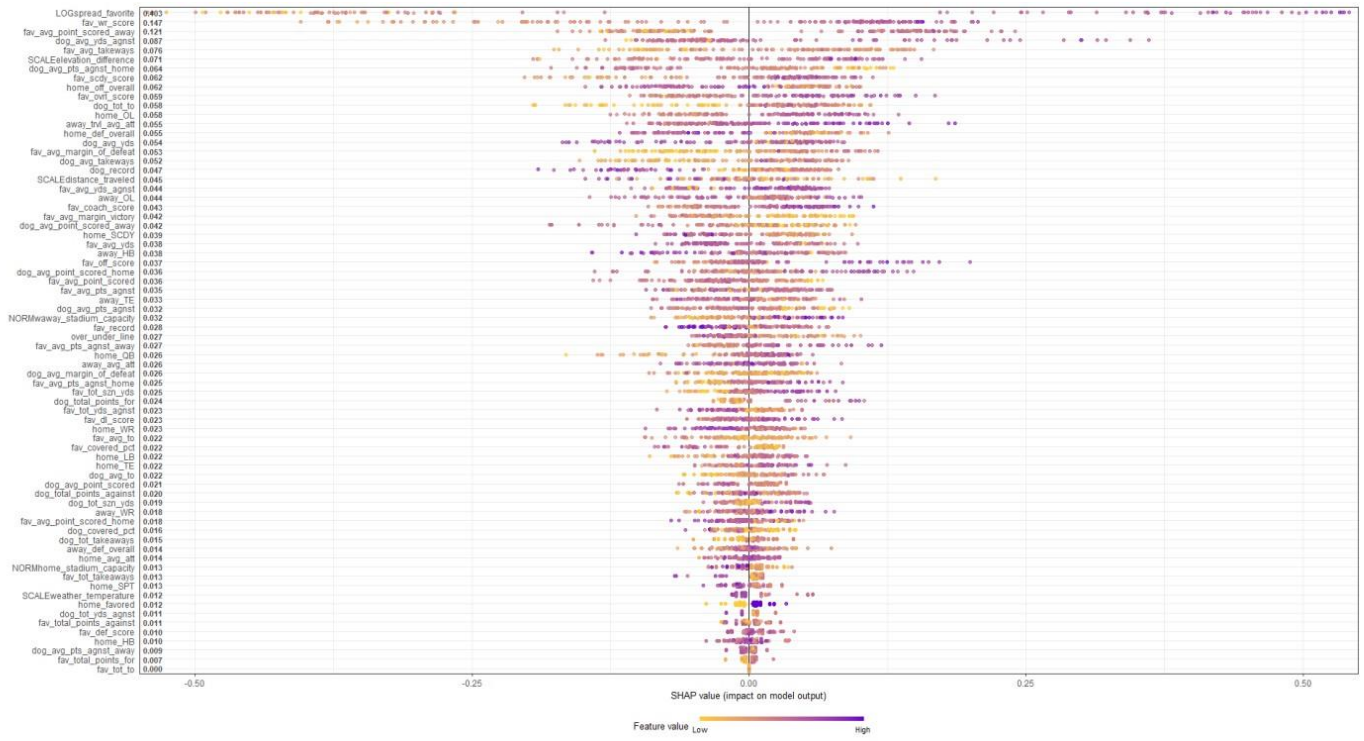


- Calculated based on
  1. Gain
- Contribution to model based on the total gain of this features splits
- Avg gain across all splits where the feature was used
- 1/2[score of new L leaf + score of new R leaf - score of original leaf] - regularization term

  2. Cover
- Number of observations related to this feature

  3. Frequency
- Percentage of the number of times a feature has been used in the trees
- The plot shows the gain on the x axis and variable on the y axis
- We can see around 12% of the total model gain was attributed to LOGspead_favorite.
- Followed by 5.4% attributed to fav_wr_score

SHAP Plots:



Y axis indicates importance with SHAP values
X axis is the SHAP value; how much is the change in log-odds

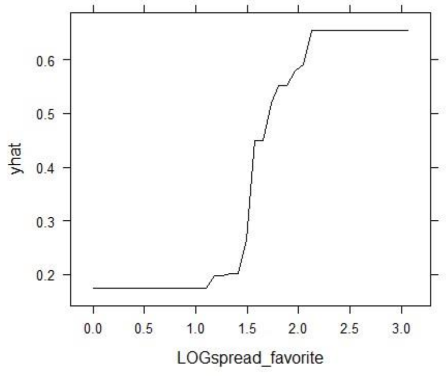
Variables: We see that if LOGspread_favorite has a high value, the SHAP value is high

- A high LOGspread_favorite value increases the chances of the favorite winning.
- A high dog_avg_yds_agnst value increases the chances of the favorite winning.
- A low fav_wr_score value decreases the chances of the favorite winning.
- A high fav_avg_points_scored_away value increases the chances of the favorite winning.

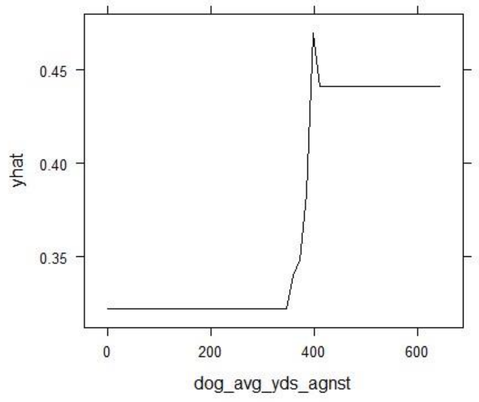
Partial Dependency Plots

- Shows the average marginal impact on the prediction with respect to a feature given value - For this example, the PDP displays the probability of [0,1] given different feature values

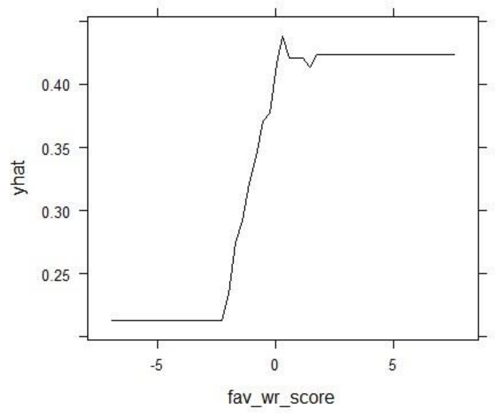
LOGspread_favorite: Once this feature hits 1.5, the probability of the favorite winning heavily increases (to >0.60)



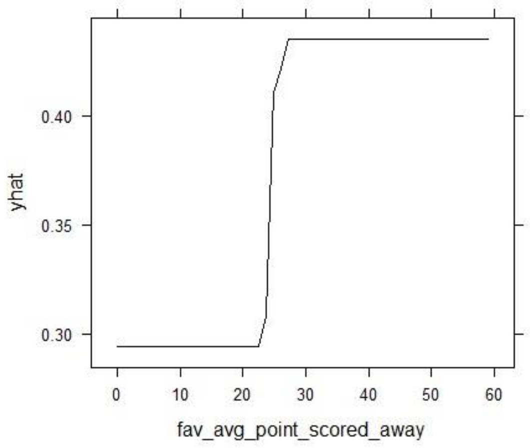
dog_avg_yds_agnst: Once this feature exceeds 375 yds, the probability of the favorite winning increases to 0.45-0.50



fav_wr_score: As this score approaches 0 (equal WR/SCDY ratings), the probability of the favorite winning begins to increase



fav_avg_points_scored_away: Once this feature reaches +21 the probability of the favorite winning spikes from 0.30 to 0.45.



In-depth summary on next page

2. In-Depth Explanation

Preprocessing

- Packages are loaded
- Data is loaded
- Data is further cleaned

Variable Selection

- Utilized a random forest to evaluate variable importance
- No need to keep variables that would further complicate the model
- Extracted Gini (Impurity)(MDG) & Mean Decrease Accuracy (MDA) Scores for each variable in the model
- Only kept variables with an MDA > 0 & MDG > 5

Steps in selecting variable importance via Random Forest

1. Fit Random Forest to the data
2. OOB Error for each data point is recorded
    - 2.2. Prediction Error via bagging
    - 2.3. Bagging - Subsampling w/replacement to create training samples
    - 2.4. Essentially the model is trained many times on each observation in its "bag" via subsampling
    - 2.5. The OOB Error is the recorded error on the observations out of the bag.
3. The feature(variable) values are permuted across the training data & OOB error is computed
    - 3.2. Values being permuted have "noise" added to them (nPerm argument)
4. The importance is then computed by averaging the difference between the OOB error before & after the permutation of all the trees.
    - 4.2. MDA = Average decrease in accuracy if x feature is removed cetibus paribus
    - 4.2 MDG = Average decrease in purity (measure of homogeneity of the predictions at the node) - Kept certain features via expert knowledge in football. (Home Being Favored)

Preprocessing II:

- The data was chronological 2016->2021
- Shuffled to eliminate bias when doing a training/testing split - Split the data 90/10.
- Turned the data into an xgb matrix obj, this is a requirement

Defining the folds:

- Split the training data into folds for cross validation
- Chose 3 folds due to size of the dataset

SCORING FUNCTION

- A scoring function that can be fed into the optimizer.
- This function will be a holder for all the parameters that will be tuned in the optimization process.

Parameters in Scoring Function:

eta:

- Rate at which model learns/updates (Learning Rate)
- eta shrinks feature weights after each boosting step
- Weights being the output of individual trees in the forest ensemble
- Shrinkage scales newly added weights after each step of tree boosting
- Shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model.
- So, the lower eta is, the lower shrinkage is applied to each weight - in turn, more learning at the expense of time/complexity

gamma:

- The minimum loss reduction required to make another split on a leaf node
- Controls the complexity of a given tree (Controls requirements (min loss reduction) to split a tree (further complicate))
- Larger values will generalize the model (Large amount of minimum loss required to make split, hence a more generalized model)

max depth:

- Maximum depth of a given tree - another complexity constraint.
- Length of longest path from root to leaf.
- Increasing allows for more complex trees & also opens the door to overfitting

min child weight:

- Minimum weight that a node needs in order to be further split - if the weight is less than the minimum, the node cannot be further split
- Helps reduce complexity & limit tree depth.
- Almost like a "purity" measure/threshold for each node that limits overfitting

subsample:

- Column (& row) subsampling. Helps reduce run time & overfitting
- Ratio of data subsampled that is used for fitting at each iteration.

alpha/lambda:

- Regularization/Penalty term on weights (L1/L2 respectively). Helps smooth final learned weights to prevent overfitting
- Larger values will make model more conservative/general by reducing weight coefficients - so smaller weights shrink
- Lambda (L2/Ridge) - adds squared magnitude as penalty to weights
- Alpha (L1/Lasso) - adds abs value of magnitude as penalty to weights

nfold:

- number of split subsamples

booster:

- Dart booster used - uses a dropout technique to prevent overfitting & remove trivial trees
    - Helps address any overfitting introduced.
- When computing the new trees gradient, only a random subset of the existing ensemble is considered - the others are dropped.

    - rate drop: The dropout rate used in the Dart booster. Can be viewed as the degree of regularization

    - normalize type: Normalization step of Dart.

Tree - New trees have same weight as dropped trees.
Forest - new trees have same weight as sum of dropped trees

Both new trees & dropped trees are attempting to achieve the same goal (optimal predictor), so to avoid overshooting the target, we can control/normalize the weights of the new trees based on the weights of the dropped trees. Using the same weights allows for consistency, using the sum allows for generalization.

sampling method:

- uniform (default): each training sample has an equal probability of being selected (other options not available on laptop(gpu_hist))

eval metric: AUC:

- chosen because of the classification threshold discussed below
- ROC summaries all the confusion matrices that each threshold produces (True Positives, False Positives)
- AUC plots all the area under the ROC curve

tree method: approx/heuristic:

- Greedy was not feasible due to computational constraints. Additionally, the quantile strategy can get the same accuracy as exact greedy given reasonable approximation level
- Approx implements splitting points based on the percentiles of feature distribution.
- Approx uses Weighted Quantile Sketch - XGB author developed the weighted version, but in essence - a quantile sketch
- Quantile sketch examines values given a desired rank. The sketch then skips over missing values & infers values based on the quantiles & ranks that were observed.
- Since the link in the original XGB paper was broke, I am assuming a weighted quantile sketch applies the same logic, but weights are introduced to the values, then the weighted values are ranked. - Broken Link in paper: Link to the supplementary material:

http://homes.cs.washington.edu/˜tqchen/pdf/xgboost-supp.pdf

OBJECTIVE FUNCTION

- Scoring function outline above was fed into the objective function and optimized using Bayesian optimization.
- The objective function is outline below
- Bayesian optimization uses a specified acquisition function to determine where to evaluate/tune the function next based on a "probabilistic belief" concerning where to evaluate next.

Acquisition Function:

- poi: Probability of improvement
- This acquisition function computes the likelihood that the function will return a result higher than the current maximum (optimization.cbe.cornell.edu)
- "Improvement" is defined as the maximum between the x's new associated value minus x's old associated value & 0.
- So, if the new associated value < the old, the maximum is negative & and formula returns a zero.
- But if we are improving, the function returns a positive, which is how much we will improve over our current best solution.
- The improvement is sampled from a normal distribution, so the point highest probability of improvement (the maximal "Improvement") is selected.

MODEL INTERPRETATION

- To interpret the model, we will investigate the hyperparameters, final variable importance, SHAP plots & partial dependency plots.

HyperParameters

eta = 0.09577576

- Higher value with respect to the bounds set, so there is more shrinkage on the weights.
  - After each boosting step we get new feature weights, a higher eta will more strongly shrink/penalize those weights causing more regularization & a more conservative model

gamma = 1.486621

- On the lower side with respect to the bounds set
- So, this adds some model complexity. Since there is less loss reduction required to make a split, there will be more available split points.
- Increases the complexity of the model

max_depth = 16

- This was the maximum bound.
- This tells us that more complexity was required for the model, since deeper trees were required for optimal tuning

min_child_weight = 14.13252

- In the middle of the set bounds.
- Helps limit tree depth to a point, but did not overly simply or complicate the model since the tuning set this in the middle

subsample = 0.5

nfold = 6

lambda = 0.6711117 & alpha = 0.5

- Lambda was set on the higher side of the bounds. Higher L2 penalty on the weights.
- Alpha was tuned to the minimal bound. Lower L1 penalty on the weights.
- Would need to do more in-depth tuning to truly tune these parameters.
- L2 may have been tuned at a higher value because it helps deal with multicollinearity in the data.

This is logical since some variables (team overall ranking) were directly related to other variables like(points scored, QB rating, etc) Also, because some variables might have been extremely insignificant (windy).

- L1 may have been tuned lower since it is a preferred regularization technique in a case with a higher number of features.

Since we already reduced the dataset, L1 may not have needed to be larger, which is why the tuning set it so low.

rate_drop = 0

- This is rather interesting
- Rate drop was set to 0 in the tuning, so we were essentially left with a standard gbtree model.
- So, in our case, no trees were dropped during the boosting process