Tyler Gourley
26 March 2025

ECEN 471 Project 3
Sanguine Synergy: Modeling Team Chemistry in Soccer

1 Introduction and Problem Formulation

1.1 Background

In soccer, success on the field is not solely determined by the individual talent of players but also by their ability to function as a cohesive unit. The concept of **team chemistry**—the interplay of familiarity, experience, and tactical synchronization among players—remains an elusive yet critical factor in predicting team performance. While traditional analytics focus on player statistics, formations, and tactical strategies, the impact of team synergy has been largely unquantifiable… until now.

This paper seeks to bridge that gap by developing a machine learning model that captures the impact of team chemistry on match outcomes. Using a dataset of playing time distributions, we construct a framework to analyze how different lineups influence goal differential (a fundamental metric of success). We will aim to solve this **regression problem** by employing techniques such as linear regression, polynomial regression, and regularization. In doing so, we explore how team chemistry among players contributes to the ultimate goal of every soccer team, coach, manager and player: winning matches.

2.2 Impact

The impact of a highly accurate model describing the effect of team chemistry would alter the landscape of soccer forever, especially if that model indicated that strong team chemistry leads to an increased probability for success. Unfortunately, using the preliminary methods that are described in this paper, we will not be able to say for certain how much impact team chemistry truly has on winning games. The goal of this paper—however—is to create a model that can accurately predict goal differential based on a team's previous playing experience together.

Even so, football coaches could use the trained model created in this paper to get a preliminary estimate as to their team's expected performance (against any random opponent) based solely on their current team synergy.

2.3 Problem Structure

The inputs to the machine learning models are 55 feature vectors where each element is quantified by the number of games two players have played together in their professional careers. Given there are 11 starters on a soccer team and are concerned about two-player combinations, this size of the input feature vector can be verified easily: ${}_{11}C_2 = 55$.

The output of the model is goal differential—represented with a single integer. That is to be calculated as the difference of goals scored and goals allowed. For this paper, the metric for goal differential will only include goals that are scored during the 90-minute period (including extra time). It will NOT include goals scored from shoot-outs or any other tie-breaking procedure. So, a match that comes down to shoot-out will always have a goal differential of 0 regardless of if a team emerges victorious due to scoring more goals in the shoot-out.

I expect that teams with high team chemistry will be more likely to perform better. Because I believe there is a strong relationship between team chemistry and team success, I am confident a model can be made that drastically outperforms a simple baseline model.

2.3 Additional Depth

This project builds upon the dataset from Projects 1 & 2. In order to go above and beyond, I will be using an increased dataset size (using data from 2008–2009).

I expect that Bagging and Boosting techniques will drastically outperform the models I have created for previous projects. Up to this point, the best model that has been the most accurate was a Decision Tree and kNN. I expect a Boosting technique will produce the most accurate model because it expands upon a relatively simple model by layering additional models on top of the base model, adding model complexity. I hypothesize that Bagging and Boosting will outperform all my previously created models because they are based off models that can accurately predict complex, nonlinear relationships. By using an ensemble of models, I presume the models created will be more accurate and have decreased error.

2 Methods and Experiments

2.1 Setup

2.1.1 Source of the Dataset

The data used in this project comes from an SQLite database that was assembled in Dr. Harrison's research lab that contains over 20 years of playing time data. Originally, the data was scraped from the ESPN website. In my responsibilities as a researcher, I was the one to compile the raw playing time data into the .csv files that will be used as the dataset for this project. I have included the dataset .csv files in the .zip file in my code submission (they can also be found at this link). We will be only analyzing data from 2008 and 2009 which contains data for 27,890 soccer matches—which corresponds to twice as many inputs (55,780) because each match has data for both teams that participated in the match.

2.1.2 Test/Train Split

First, we separated the data into a training dataset and test dataset. The nature of the dataset is that each match has 2 inputs associated with it (one for each team). So, in order to avoid data leakage into the test set, the data was first grouped by match before performing a test/train split. We do this by using the GroupShuffleSplit() method splitting on match id (see documentation for sklearn.model_selection.GroupShuffleSplit). By stratifying the random split by match id, we are able to ensure that inputs from the same match are both found in the same set (either both in the test set or both in the training set). We used a 90/10 split—90% for training data and 10% for test data.

2.1.3 Estimating Enew

We will use the holdout validation method to estimate the error on data the model has never seen before (E_{new}). This decision was made to decrease the computation time of estimating E_{new}—holdout validation is less computationally expensive than cross-validation. Because we have a large dataset, we expect this method will provide accurate estimates for E_{new}.

When creating the holdout validation set, we also stratified the data by match id in order to prevent data leakage (see section 2.2). We decided to use 20% of the *original* dataset as the validation set. Because we had already split the data into train and test, we were sure to take that into account and split the data accordingly. We normalized the train/validation split to the 20/90 ≈ 0.22 (~22.2% validate and ~77.8% training). This left the final allocation of the data as 70% training data, 20% validation data, and 10% test data.

We set the random seed to be 999 for all functions that use randomness.

2.1.4 Error Function

The nature of our output data (goal differential) is numerical. Thus, we will be using regression models. The error function we will be optimizing for is mean squared error (MSE).

2.2 Base Model

2.2.1 Baseline Model

The first step is to train a base model. This base model will have 2 purposes. It will (1) be the baseline to which we compare the other Bagging and Boosting models and (2) serve as the input model for the bagging algorithm. I chose to use kNN for the baseline model. I chose kNN because I learned from Project 1 that kNN performs very well on my dataset. I used the Scikit-learn kNN Regressor (see sklearn.neighbors.KNeighborsRegressor) to create the baseline model.

For training a k-Nearest Neighbors (kNN) model, we must determine what value of k (number of closest neighbors) yields the most accurate results. In order to do this, I first performed a hyperparameter sweep over the following list:

k: [1, 31, 75, 123, 149, 179, 280, 401, 597]

We used the holdout set to estimate E_{new} of each hyperparameter. At the same time, we calculated E_{train} for each hyperparameter. We plotted those results (see Figure 2.2.1). While difficult to tell on the graph alone, we found that the lowest error occurred with the hyperparameter of k=123, which produced an estimated error on new data of 2.8877. I did this to narrow down the approximate range for best performing values of k.

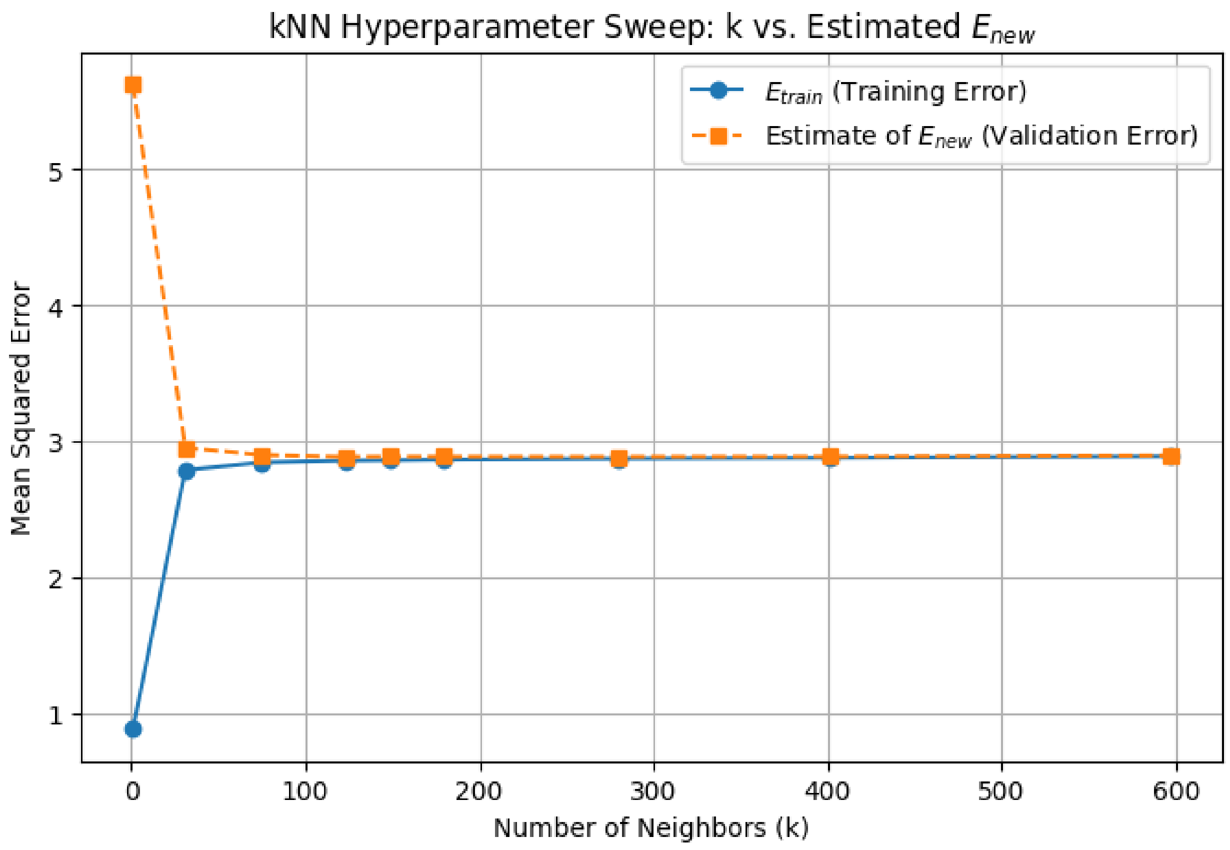


Figure 2.2.1 – kNN Hyperparameter Sweep to determine baseline model (Neighbors vs. Error)

After determining that the lowest estimated error on new data occurred with k=123, my next step was to narrow down the parameter search to those values which are close to 123. I performed another hyperparameter sweep over the following list:

k: [111, 117, 121, 122, 123, 124, 127, 133, 137]

We used the holdout set to estimate E_{new} of each hyperparameter. At the same time, we calculated E_{train} for each hyperparameter. We found that the best value for the hyperparameter k to be 124, which produced an estimated E_{new} of 2.8875. We plotted those results (see Figure 2.2.2). At this zoomed-in range of data, we are able to more clearly see what looks like an inflection point of E_{new} at k=124. We can now be fairly confident that 124 is the best option of the number of neighbors hyperparameter for our kNN model.

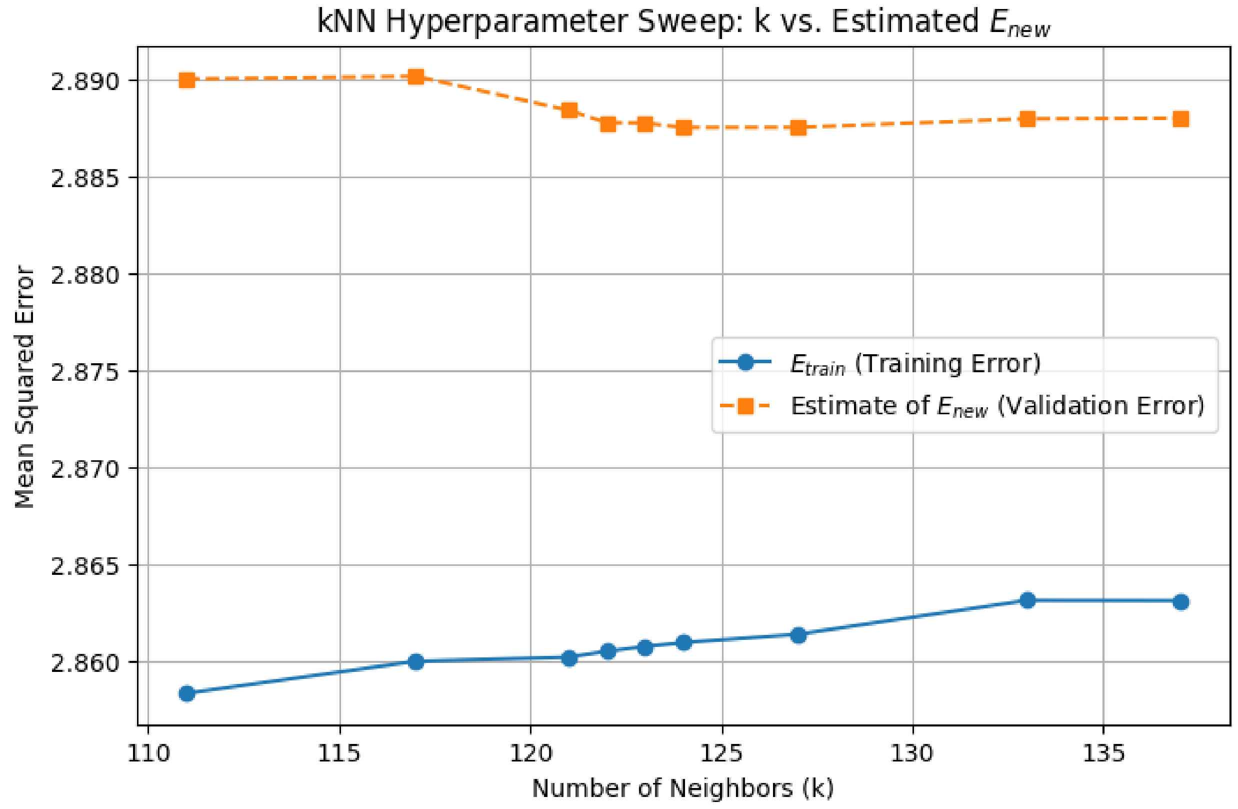


Figure 2.2.2 – kNN Hyperparameter Sweep to determine baseline model (Neighbors vs. Error)

Finally, we trained a kNN model on the full dataset (comprising of both the training set and validation set, but not including the test set) using 124 as our value for k.

2.3 Bagging Estimator

2.3.1 Determine the Best Bagging Estimator

Bagging is a method used to make machine learning models more stable and accurate. It works by using a process called bootstrapping, where random samples of the training data are put into many bags (sampling is done with replacement, so some data points can even appear in the same bag more than once). Each bag of samples is used to train a separate model. For regression problems, the final prediction is the average of all models' results. This approach helps reduce error and improves accuracy by reducing variance.

Now we look to determine the best bagging estimator. We will use the scikit-learn Bagging model for regression (see sklearn.ensemble.BaggingRegressor). We will use the best baseline kNN model that we determined to be k=124 (see Section 2.2). For the other model parameters (number of estimators, maximum samples, and maximum features) we will have to sweep over many values to determine the best combination. We performed a hyperparameter sweep over the following values:

n_estimators: [10, 50, 100]

max_samples: [0.5, 0.75, 0.9]

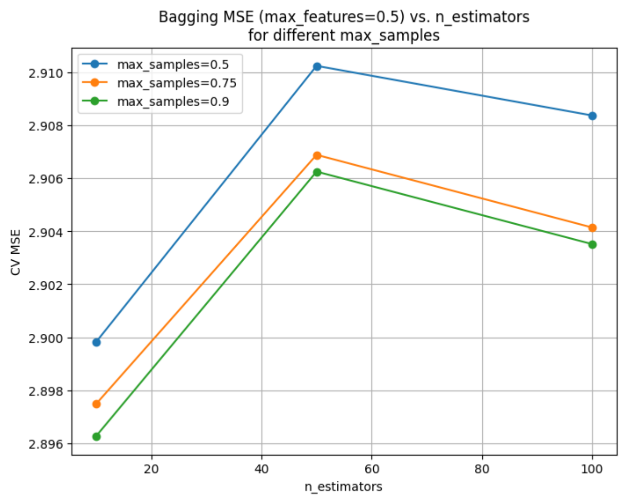
max_features: [0.5, 0.75, 0.9]

With each combination of hyperparameters, we used the holdout validation strategy to estimate E_{new}. We graphed the results below (Figure 2.3.1 A–C). This hyperparameter sweep took over an hour and a half to complete on a Google Colab Notebook. In the future, I would instead shorten the list of hyperparameters and do a more sophisticated parameter search (similar to what was done in Section 2.2 to create the kNN baseline model). Overall, we determined that the best combination of hyperparameters for the Bagging model is as follows (producing an expected E_{new} of 2.8857):

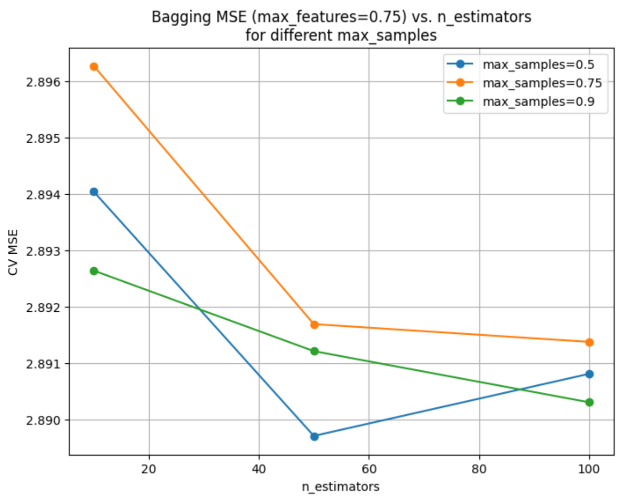
n_estimators = 50

max_samples = 0.5

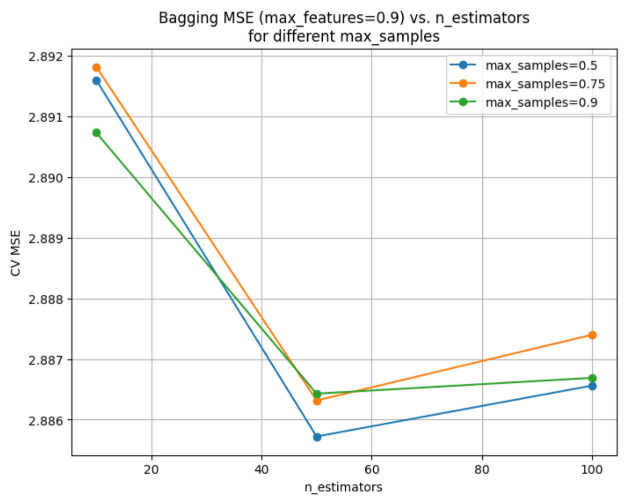
max_features = 0.9



(A)



(B)



(C)

Figure 2.3.1 A–C – Bagging Hyperparameter Sweep (Bagging MSE vs. Number of Estimators)

Finally, we trained a final bagging model (using n_estimators=50, max_samples=0.5, and max_features=0.9) trained using both the train and validation sets.

2.4 Model Using Boosting

2.4.1 Determine the Best Boosting Estimator

Boosting is a strategy to create improved machine learning models by combining several weaker models one after another. In gradient boosting, each new model is trained on the gradient of the error of the previous model, with the goal of fixing the mistakes made by the ones before it. By learning from these mistakes, accuracy increases with depth, making it good for handling complicated data and reducing bias (by increasing model complexity).

Now we look to determine the best boosting estimator. We will use the XGBoost boosting model for regression (see xgboost.XGBRegressor). We will use the best baseline kNN model that we determined to be k=124 (see Section 2.2). For the other model parameters (number of estimators, learning rate, and maximum depth) we will have to test different values to determine the best combination. We performed a hyperparameter sweep over the following values:

n_estimators: [10, 50, 100, 200, 350]

learning_rate: [0.01, 0.05, 0.1, 0.2]

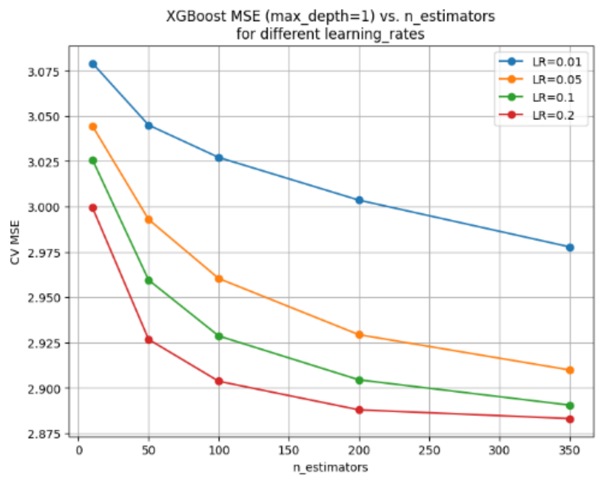
max_depth: [1, 2, 3, 5, 7, 10]

With each combination of hyperparameters, we used the holdout validation strategy to estimate E_{new}. We graphed the results below (Figure 2.4.1 A–F). The lowest MSE model produced an error of 2.8733 on the holdout validation dataset. We determined that the best combination of hyperparameters for the boosting model is as follows:

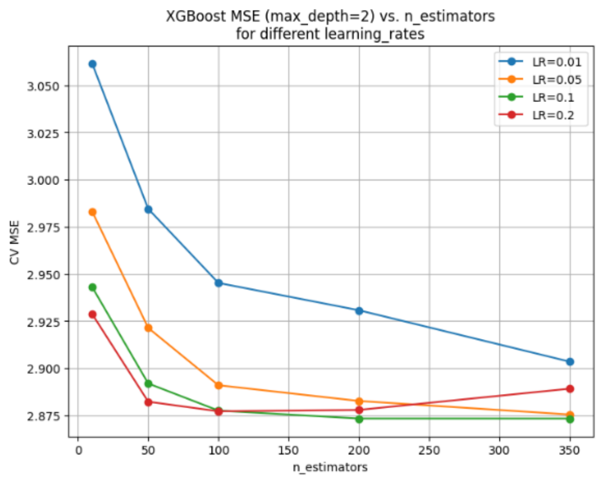
n_estimators = 350

learning_rate = 0.1

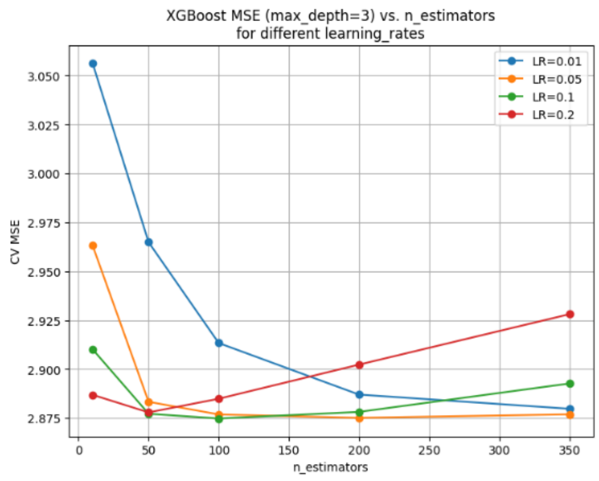
max_ depth = 2



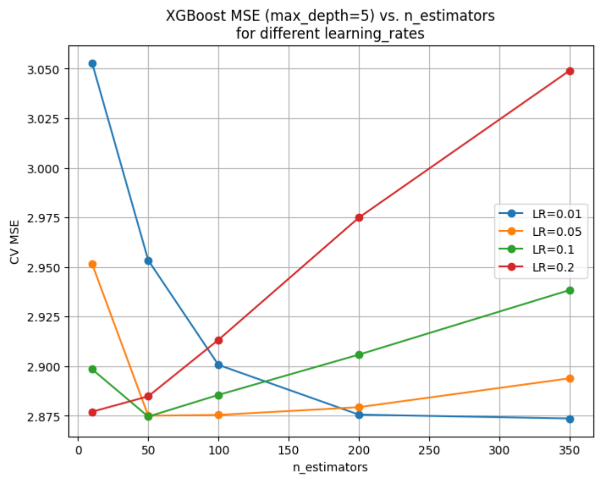
(A)



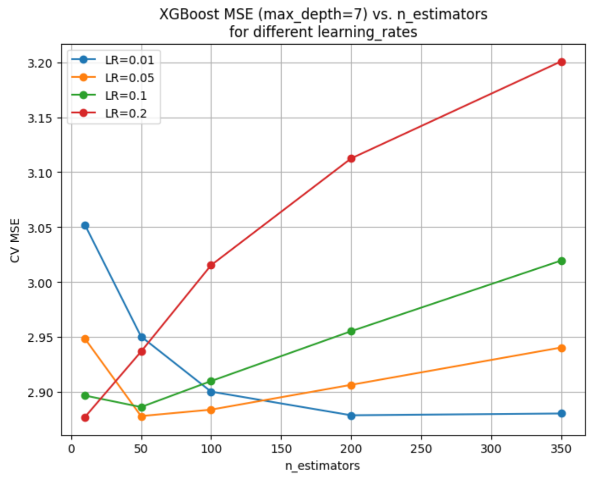
(B)

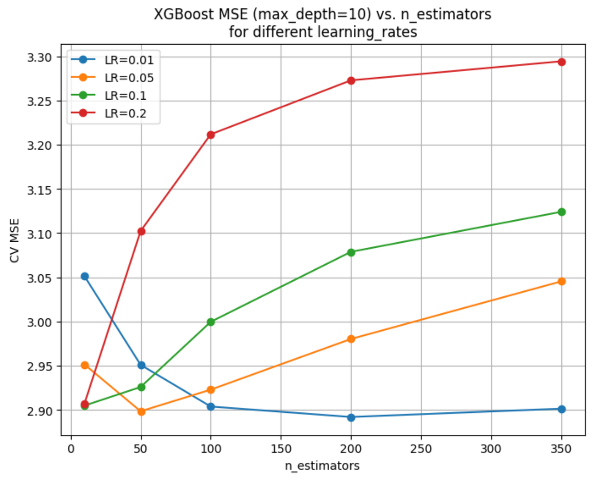


(C)



(D)





(E) *(F)*

Figure 2.4.1 – Boosting Hyperparameter Sweep (Boosting MSE vs. Number of Estimators)

Finally, we trained a final boosting model using the best parameters as determined by our hyperparameter sweep, trained on both the train and validation datasets.

2.5 Final Testing and Comparison

2.5.1 Test all models on Test dataset

For the 9 final models we have trained, we used the test dataset to evaluate each model's performance. Here are the results:

Model	E_{test} (MSE)
Baseline (kNN w/ k=124)	2.8761
Bagging	2.8643
Boosting	2.8664

Table 2.5.1 – Table of final performances (E_{test}) for each of the final models

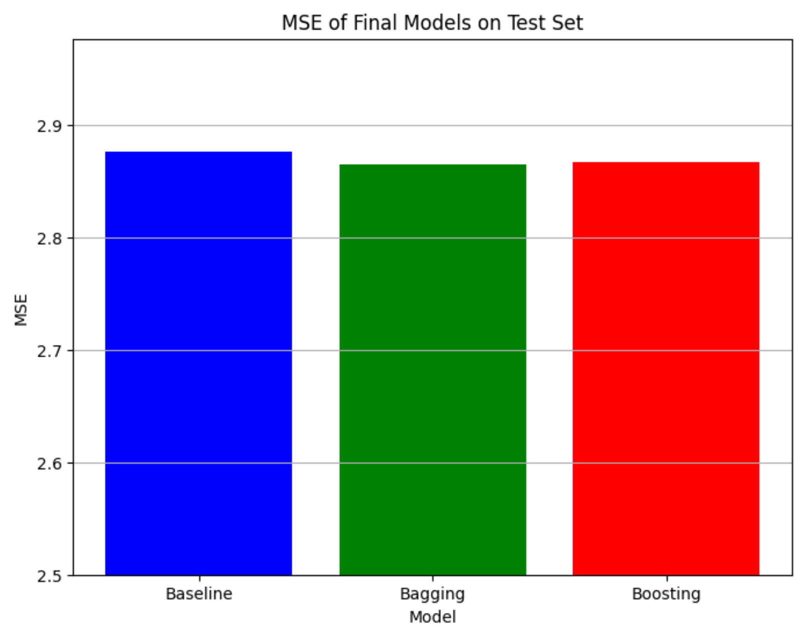


Figure 2.5.1 – Plot of E_{test} for each of the final models

3 Discussion and Conclusion

3.1 Analyze Performance Metrics

We can see that the Bagging technique produced the best model (with the lowest error on the test dataset). However, both bagging and boosting only slightly improve upon the Baseline kNN model. Frankly, this is a disappointing result to see less than a 1% decrease in error compared to the kNN model with k=124.

3.2 Discussion and Final Conclusion

Unfortunately, I do not have much confidence in any of the models that were created in this project. If the models I create continue to only slightly outperform a simple “predict 0” model, I will also have to start facing the reality that playing time may not be a great predictor of success for soccer. There are a lot of limitations with the models I have created, given that they do not perform much better than random guessing. I would not feel confident using any of these models in the real world. I have concluded that Bagging and Boosting with kNN as the baseline is not the way to go. In the future I may look into using a Decision Tree with boosting and bagging algorithms.

Compared to the models that were created from Projects 1 and 2, these models actually perform worse than the kNN and Decision Tree models from Project 1. This is NOT a fair comparison, however, because I am using a larger dataset in this project (using more matches over a longer period of time) than I used in Project 1. I am confident that if I did Project 1 over again using the expanded dataset, Bagging and Boosting would outperform standalone kNN and Decision Tree models. In the future, I look forward to using Convolutional Neural Networks and Clustering

Algorithms to experiment with more complex models. I expect that a CNN will work best because it will be able to capture the complexity of playing time data housed in my model.