Tyler Gourley
5 February 2025

ECEN 471 Project 1: 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** (see section 2.3) by employing techniques such as decision trees and k-nearest neighbors. 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.

2.2 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 Methods and Experiments

2.1 Source of 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 a .csv file that will be used as the dataset for this project. We will be only analyzing data from the year 2008 which contains data for 13,507 soccer matches—which corresponds to twice as many inputs (27,014) because each match has data for both teams that participated in the match.

2.2 Perform a Train/Test split on the dataset

Data was separated 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. 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.

2.3 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).

2.4 Estimating E_{new}

We will be estimating the mean squared error on data the model has never seen before (E_{new}) using the holdout validation method. 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 will set the random seed to be 42 for all functions that use randomness. Additionally, even though the standard convention is to scale the input data (often with a Standard Scalar or another similar function) before training the model, I decided against it for this project. This is because I predict that exceptionally high playing time values will be a strong indicator of a large goal differential… so I am not worried about those large input values skewing the output.

2.5 Baseline Model

We will use Scikit-learn Dummy Regressor (from the sklearn.dummy.DummyRegressor package) as our baseline model to which we will compare the machine learning models we create. The strategy of the baseline model was set to always estimate the statistical mean of the training set. This was accomplished by setting the strategy parameter to 'mean' (strategy='mean'). All other parameters were left to the default values.

Using the holdout validation strategy on the baseline model, we get $E_{new} \approx 3.0640$ for MSE.

Finally, we trained a model on the full dataset (comprising of both the training set and validation set, but not including the test set) to use as our baseline model for the performance evaluation of the final models.

2.6 k-Nearest Neighbors Algorithm

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

k: [1, 3, 5, 7, 9, 15, 21, 29, 41, 55, 69, 91, 111, 123, 149, 163, 179, 199, 251, 301, 395, 479, 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 2.6.1*). While difficult to tell on the graph alone, we found that the lowest error occurred with the hyperparameter of k=163, which produced an estimated E_{new} of 2.9009.

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 163 as our value for k.

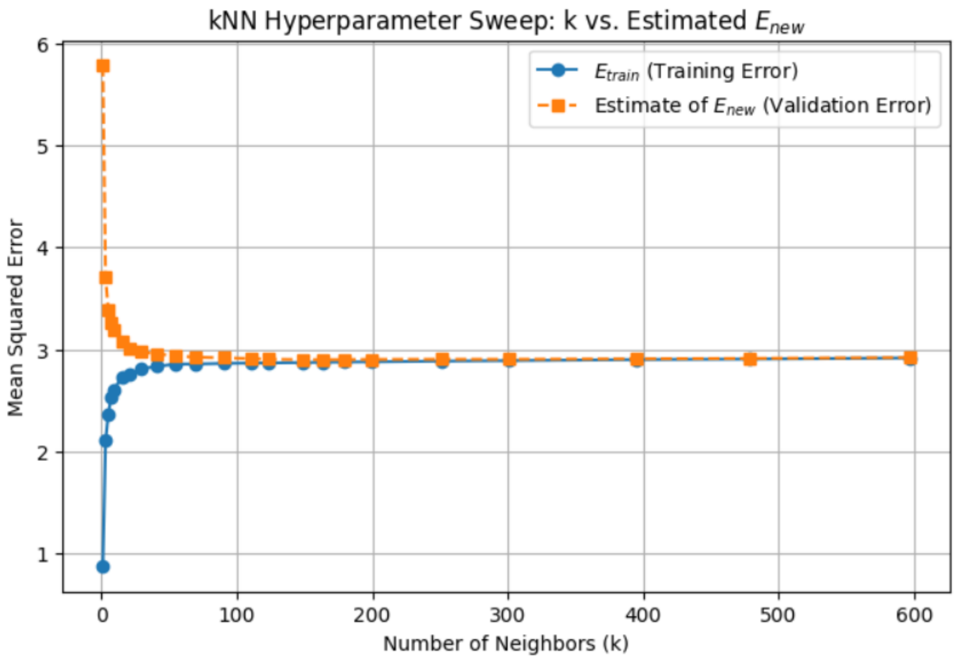


Figure 2.6.1: Estimate of E_{new} *using holdout validation and* E_{train} *of the kNN parameter sweep*

2.7 Decision Tree Algorithm

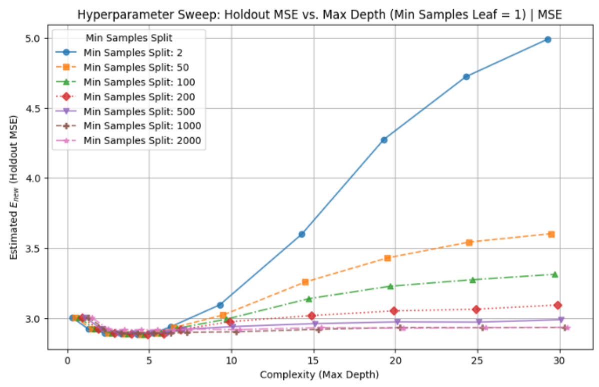
For decision tree regression models, there are four criterion loss functions that determine the way regions are split. We chose to create models with two criterion loss functions: 'squared_error' and 'friedman_mse.' We chose three parameters to sweep over: max depth, min samples split, and min samples leaf. We set the random_state parameter to always be 42, and let every other parameter be the default. For each criterion loss function, we will train a decision tree model and calculate the MSE for every possible combination of these hyperparameters:

'max_depth': [1, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25, 30],

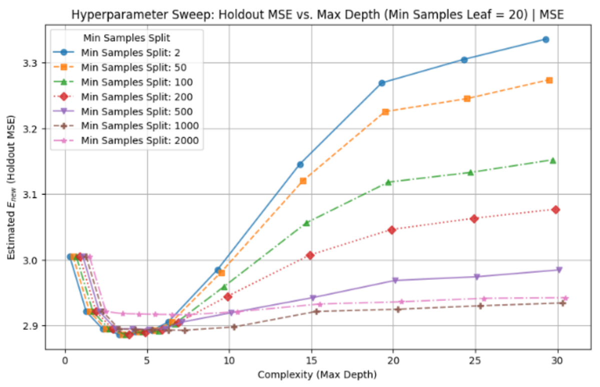
'min_samples_split': [2, 50, 100, 200, 500, 1000, 2000],

'min_samples_leaf': [1, 20, 50, 100, 250, 500]

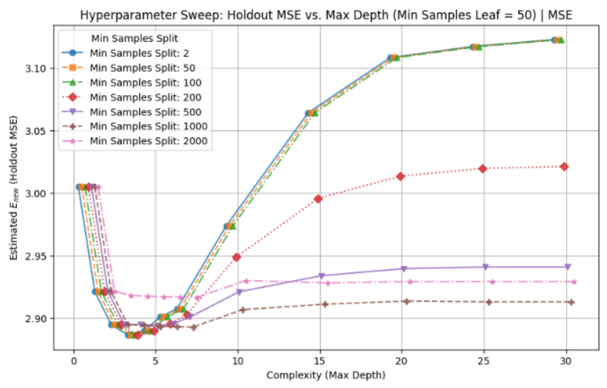
We used the holdout set to estimate the value of E_{new} for each hyperparameter combination. The series of plots below show the results of this hyperparameter sweep. *Figure 2.7.1* shows the hyperparameter sweep for the 'squared_error' criterion, while *Figure 2.7.2* shows it for the 'absolute error' criterion. We see from these graphs that they largely resemble the Error vs. Model Complexity graph discussed in class (lovingly nick-named the 'Nike Swoosh' graph). We see a clear inflection point, or 'sweet spot,' where the E_{new} estimate is at its lowest.



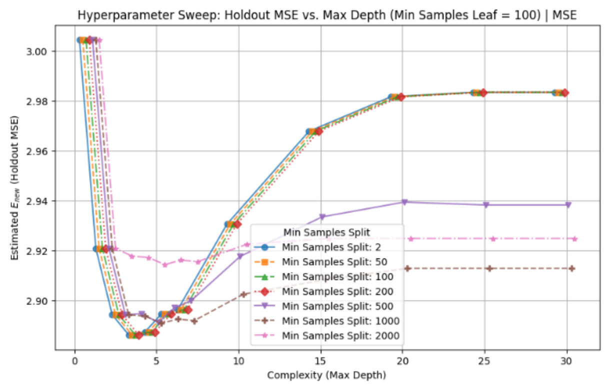
(a)



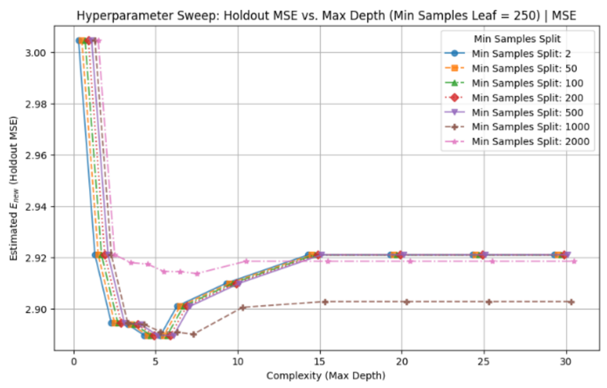
(b)



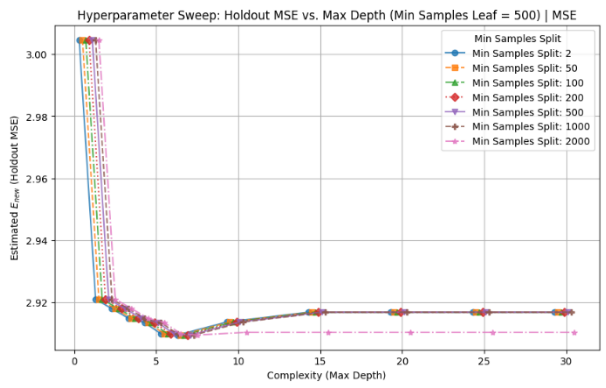
(c)



(d)

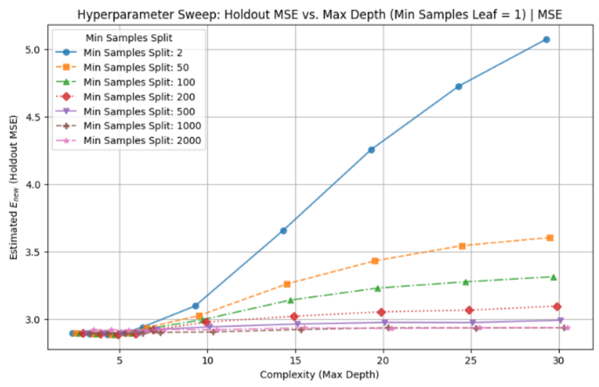


(e)

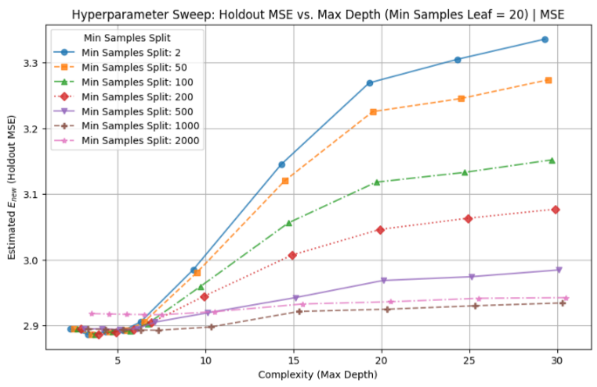


(f)

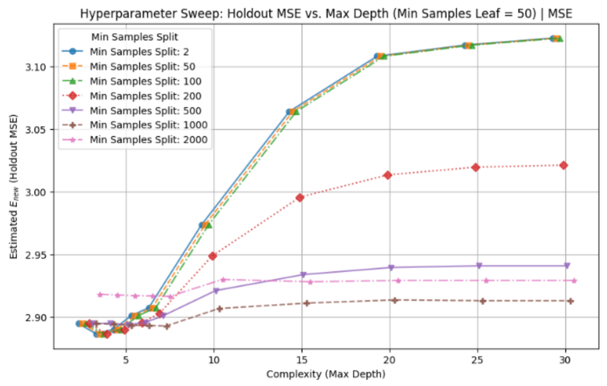
Figure 2.7.1: Decision tree hyperparameter sweep for the 'squared_error' criterion



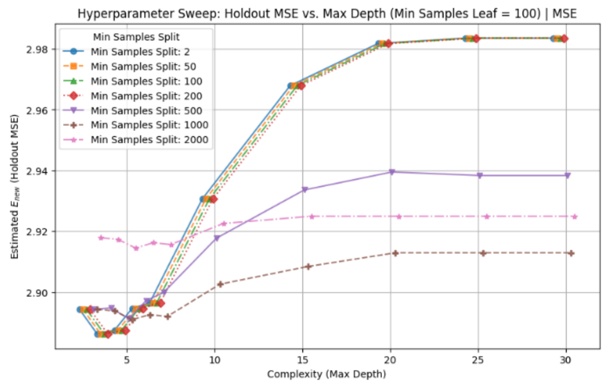
(a)



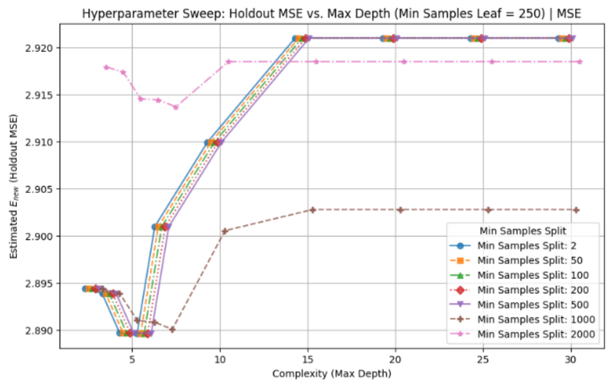
(b)



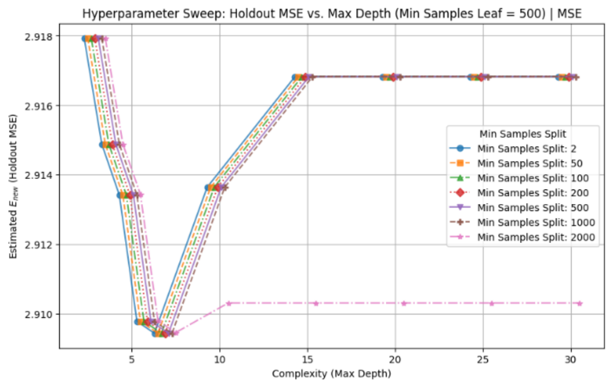
(c)



(d)



(e)



(f)

Figure 2.7.2: Decision tree hyperparameter sweep for the 'friedman *_mse' criterion*

After performing the sweep over the hyperparameters, we found the best parameters for each criterion. For the 'squared_error' criterion, we determined that the best model (the one with the lowest estimated E_{new}) occurred with the following hyperparameters:

max_depth=5, min_samples_split=2, and min_samples_leaf=1

Using the holdout dataset to compute the MSE, this model had an estimated E_{new} of 2.8848.

For the ‘friedman_mse’ criterion, we determined that the best model had the same hyperparameters E_{new} estimate as the ‘squared_error’ criterion. Upon further inspection of the sklearn documentation, I found that the ‘friedman_mse’ uses the ‘squared_error’ criterion as its base and adds a slight adjustment called a Friedman improvement score.

The MSE of the ‘friedman_mse’—again using the holdout validation set—yielded an estimate for E_{new} of 2.8848.

Finally, we trained a decision tree model on the full dataset (not including the test set). Because the errors were the same for both squared error and friedman mse, I just chose to use criterion=‘squared_error’ which left us with a model using the following parameters (all other parameters were left to the default values):

‘random_state’=42, criterion=‘squared_error’, max_depth=5, min_samples_split=2, and min_samples_leaf=1

2.8 Evaluate Performance

Now that we have tuned, created, and trained 3 models (Baseline, kNN, and Decision Tree), we are able to evaluate the performance. We will do this by using the test set that we set aside initially and calculating the Mean Squared Error.

Model	Mean Squared Error
Baseline Model	2.9045
k-Nearest Neighbors Model	2.7321
Decision Tree Model (squared_error)	2.7235

Table 2.8.1: Mean Squared Error on Test set for each final model

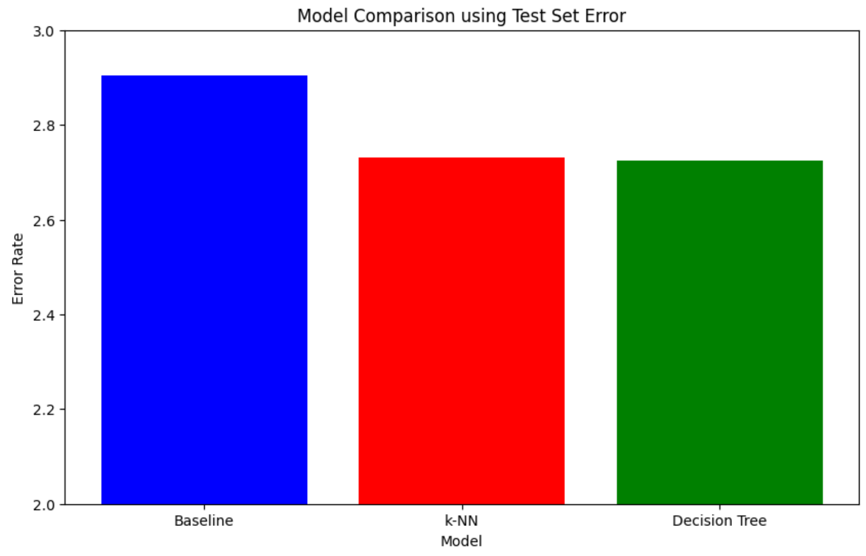


Figure 2.8.2: Plot of Mean Squared Error on Test set for each final model

3 Discussion and Conclusion

3.1 Analyze Performance Metrics

The model that performed the best was the decision tree model using the squared_error criterion. I came to this conclusion because this model had the lowest E_{new} error on the test set. Granted, both the kNN and the decision tree model are very similar, so either one can be considered best.

3.2 Comparison to Baseline Model

The decision tree model outperformed the baseline model, but only slightly. In the future, I would like to train models using Decision Tree and k-Nearest Neighbor algorithms as Classification models instead of Regression models. This would necessitate updating the labels from a numerical variable (goal differential) to a categorical variable (e.g. Win, Loss, Tie). Additionally, I would like to look at whether or not team chemistry can predict an advantage a team may have over their opponent. One way to go about this would be to combine both teams' inputs from a certain match into a single input.

3.3 Discussion and Final Conclusion

Unfortunately, I do not have much confidence in the decision tree model. However, there is no negative impact of failure for this model. There will be (on average) a 1.65 difference between the predicted goal differential of the model and the actual outcome of the game. So, at that point, I would not lend too much credence to the model, but I would still use the decision tree model for a preliminary prediction.