Huddle: A Machine Learning application for Team Recommendation

Akshay Jain
Master of Science in Software Engineering, ASU
ajain122@asu.edu

Ishan Shrivastava
Master of Computer Science, ASU
ishrivas@asu.edu

Kunal Sutradhar
Master of Computer Science, ASU
ksutradh@asu.edu

Meha Shah
Master of Computer Science, ASU
mshah17@asu.edu

Nikita Shanker
Master of Computer Science, ASU
nshanker@asu.edu

Radha Kannan
Master of Computer Science, ASU
rkanna11@asu.edu

Table of Contents

ABSTRACT

The prevalence of fantasy sports has been multiplying in recent years all over the globe. Fantasy Football for English Premier League is one of those fantasy sports gaining popularity among the young football fans. Fantasy Premier League (FPL) is a worldwide online platform that allows users who want to be a part of FPL to choose their own team for a certain game week. Official site for FPL (https://fantasy.premierleague.com/) offers week after week game play where individuals pay to form a dream team of 15 players under some money constraint looking for the opportunity to win cash. While machine learning has been put into use for fantasy sports predictions, very little efforts have been made to come up with concrete team selection strategies for FPL. This paper illustrates how such decisions can be made by utilizing machine learning procedures and develop models by identifying suitable factors. Certain techniques such as different classification algorithms, time-series prediction analysis, random walk graph kernel algorithms are explored using data for 3 FPL seasons from several data sources.

Keywords

Fantasy Football, Dream team, Time-Series, Ensemble model, FPL Data, Player Replacement

1. INTRODUCTION

In the past two decades, the internet has exponentially grown, resulting in the growth of online platforms for fantasy sports. Fantasy sports allow diehard sports fans to express support for real players all over the world. Large number of users have been using multiple betting sites where they select their set of players and compete against other people for cash awards. The main challenge is to select players which provide good statistical performance relative to their price in the draft. The purpose of this project is to create models to facilitate player selection decision-making for FPL players, and player replacement strategies for a football team manager. We set out to build statistical models that predicts team/player performance based on rich historical data, after identifying most important features to base our models on.

A lot of strategic planning and tactical thinking goes behind selecting players by FPL users. Since an FPL player has a budget constraint of £100m to spend on 15 players in a team, it gets very hard to choose players for a certain game week. We felt that machine learning can ease up the process of making decisions for selecting a team that scores maximum points in a week. We also deal with another aspect of the team selection/recommendation for a football club/team manager focused on suggesting replacement for an injured player or a bad performing player. Former aspect is modeled using classification models, and time-series prediction analysis, while the latter is modeled using random walk based-graph kernel algorithm.

For the purpose of predicting the performance of players, the data for individual player performance were collected from various sources mentioned in the references section of this report. The data collected includes goal scored, points of the last round, bonus, saves, red cards, yellow cards, clean sheets, assists, etc.

Paper Organization

The following sections of this paper are organized as follows:

- Section 2 provides a description of the problem identified, and approaches used to build machine learning models and the specifics of the implementation techniques used.
- Section 3 presents detailed theoretical analysis of all the algorithms used to build models for player selection/replacement.
- Section 4 provides the final detailed results for the project including the possible reflections for all the approaches.
- Section 5 contains the conclusion, a brief summary of the main contributions of all the team members, a list of some potential future work, and sources referenced throughout the project.

2. DESCRIPTION

2.1 Problem Formulation

The problem we address through this project is to build statistical machine learning models to predict high performing players for composing a high-performance football team. This problem is broken down for two kinds of people, one is the FPL player, and the other is the football club/team manager.

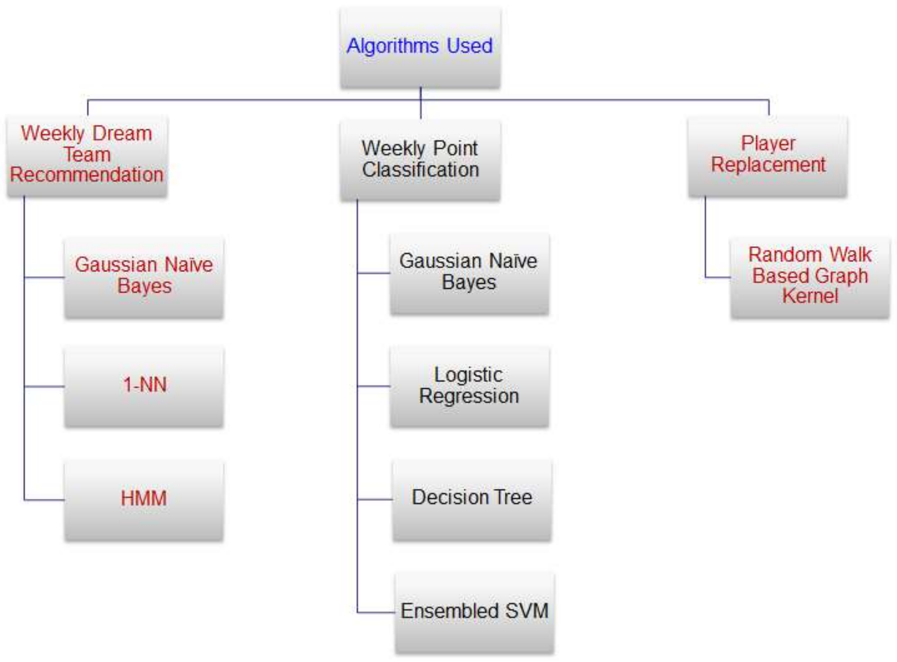
- An FPL player needs to decide the football (EPL) players for their FPL team, and
- a manager needs to choose a player for replacement for a player on the field.

Given the data for Fantasy English Premier League, we aim to achieve the following three tasks that aid the FPL users and the football club managers in their decision-making process:

- If a player will score good points in the upcoming week, points greater than or equal to seven (>= 7) are considered good point score.
- Whether a player will be a part of the dream team, dream team is the team with highest performing player released by the Fantasy Premier League.
- Finding a replacement for a player on field from the players among those on the bench, replacement can be in case of injury, or a bad performing player, etc.

2.2 Algorithms/Models Used

This section describes the algorithms used for the three problems that we are tackling in this paper.



2.2.1. *Hidden Markov Model*

The HMM is a sequence model. Its job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. Given a sequence of units, it computes the likelihood of the sequence of labels and chooses the best label sequence. It allows us to talk about observed events (like the total number of points scored by a player in FPL every game week, that we see in the input) and hidden events (like if the player would feature in that game weeks FPL dream team). An HMM is specified by the following components:

$Q = q_1\ q_2 \ldots q_N$	A set of 'N' states
$A = a_{11}\ a_{12} \ldots a_{n1} \ldots a_{nn}$	A transition probability matrix 'A'; a_{ij} represents the probability of moving from state i to j, s.t.
$O = o_1\ o_2 \ldots o_T$	A sequence of 'T' observations
$B = b_i(o_t)$	A sequence of observation likelihoods, also called as emission probabilities, each expressing the probability of an observation o_t being generated from a state i
q_0, q_F	A special start and end (final) state

It is a technique borrowed by machine learning from the field of statistics. It is a go to method for binary classification. It is named after the logistic function (a function used at the core of its method). In Logistic Regression, the input values (x) are combined linearly using weights or coefficient values to predict an output value (y). The output value being modeled is a binary value (0 or 1). Each column in our input data will be associated with a coefficient (a constant real value) that must be learned from our training data. The actual representation of the model is the coefficients of each column.

2.2.2. Logistic Regression

It is a technique borrowed by machine learning from the field of statistics. It is a go to method for binary classification. It is named after the logistic function (a function used at the core of its method). In Logistic Regression, the input values (x) are combined linearly using weights or coefficient values to predict an output value (y). The output value being modeled is a binary value (0 or 1). Each column in our input data will be associated with a coefficient (a constant real value) that must be learned from our training data. The actual representation of the model is the coefficients of each column.

Logistic Regression models the probability of the default class. We model the probability that an input (x) belongs to the default class (y = 1). Logistic Regression is a linear method, but the predictions are transformed using the logistic function. The coefficients are estimated using maximum likelihood estimation. The best coefficients would result in a model that would predict a value very close to 1 for the default class and a value very close to 0 for the other class. The intuition for Maximum Likelihood Estimation for Logistic Regression is that a search procedure seeks values for the coefficient that minimize the error in the probabilities predicted by the model to those in the data. A minimization algorithm is used to optimize the best values for the coefficients of our training data.

Assumptions made by Logistic Regression:

Logistic Regression is intended for binary (two class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.

- Logistic regression assumes no error in the output variable (y).
- Logistic regression is a linear algorithm (with a nonlinear transform on output). It does assume a linear relationship between the input variables with the output.
- The model can overfit if you have multiple highly correlated inputs.
- It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many correlated inputs in your data.

2.2.3. Decision Tree

A decision tree classifies data items by posing a series of questions about the features associated with the items. Each question is contained in a node, and every internal node points to one child node for each possible answer to its question. The questions thereby form a hierarchy, encoded as a tree. In the simplest form, we ask yes-or-no questions, and each internal node has a ‘yes’ child and a ‘no’ child. An item is sorted into a class by following the path from the topmost node, the root, to a node without children, a leaf, according to the answers that apply to the item under consideration. An item is assigned to the class that has been associated with the leaf it reaches. In some variations, each leaf contains a probability distribution over the class that estimates the conditional probability that an item reaching the leaf belongs to a given class. Nonetheless, estimation of unbiased probabilities can be difficult.

2.2.4. Support Vector Machines

It performs classification by finding the hyperplane that maximizes the margin between the two classes. The vectors that define the hyperplane are the support vectors. To define an optimal hyperplane, we need to maximize the width of the margin. The beauty of SVM is that if the data is linearly separable, there is a unique global minimum value. An ideal SVM analysis should produce a hyperplane that completely separates the vectors into two non-overlapping classes. However, perfect separation may not be possible, or it may result in a model with so many cases that the model does not classify correctly. In this situation SVM finds the hyperplane that maximizes the margin and minimizes the misclassifications. The algorithm tries to maintain the slack variable to zero while maximizing margin. However, it does not minimize the number of misclassifications but the sum of distances from the margin hyperplanes. SVM handles nonlinear regions by using a kernel function to map the data into a different space where a hyperplane cannot be used to do the separation. It means a nonlinear function is learned by a linear learning machine in a high dimensional feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space. This is called kernel trick which means the kernel function transforms the data into a higher dimensional feature space to make it possible to perform the linear separation.

2.2.5. Gaussian Naive Bayes

Naive Bayes is a machine learning classification algorithm. It will assign the class labels taken from the finite set to the test data set. As per the Naive Bayes algorithm, the value of one feature is independent of the value of the other feature. Each of the feature will contribute individually to the probability that a particular test data will correspond to the particular class label regardless of any possible correlation between the features. One of the advantage of naive Bayes algorithm is that it only requires a small number of training data to estimate the parameters necessary for classification. To deal with the continuous data, we will use Gaussian Naive Bayes where we will assume that the continuous values with respect to each class are distributed according to Gaussian distribution.

2.2.6. K Nearest Neighbor

k-NN is the lazy learning algorithm, where all the computation is implemented after classification. The training set will contain vectors in a multidimensional feature space each with a class label. In the training phase, the feature vectors and class labels of the training samples will be stored. Then in the classification phase, an unlabeled vector (test data point) will be classified by using the value of k given by the user. Here, k is the number of nearest neighbors which will help the model to classify the test data point by assigning the label which is most frequent among the k training samples nearest to that test data point. A commonly used distance metric to calculate the distance of unlabeled test data point from the training data points is Euclidean distance. A drawback of this majority voting classification occurs when the class distribution is biased. That is, data points of the more frequent class will most likely dominate the prediction of the new example, because they will be very much common among the k nearest neighbors due to their large number. One way to overcome this problem is by giving weights to the classification based on the distance of the test point from each of its k nearest neighbors. The class of each of the k nearest points will then be multiplied by a weight proportional to the inverse of the distance from that data point to the test data point.

2.2.7. Random Walk Graph Kernel

The Player recommendation algorithm is based on the Fast Algorithm for Team Member Replacement [1]. In our model, we address the problem often faced by the Team Manager, of finding a good replacement, in case of a player's absence. In the crunch hours of an ongoing match, a replaced player can make or break the game.

Research has shown that team members perform well when they share familiarity with each other, both communication wise and experience wise. For example, a defender cannot replace a forward on any match day. Further, decisions have to be made about the difference in team structure. So, it is highly likely that a disconnected player will cause a greater imbalance in the newly structured team.

Our aim is to ensure individual skill matching as well as overall team preservation to minimize the impact of a player departure. Most of the other existing techniques available currently do not measure similarity in context of both our problem objectives. Hence, we have used graph kernels (as in mentioned paper) to measure the interactions between individual skillsets and team structural matching.

2.2.8. Usage of Random Walk Graph kernels

Graph kernels are used to compute the similarity between two graphs by comparing substructures of graphs. Out of the many graph kernels available such as Sub-Tree Graph Kernels, Shortest-Path Graph Kernels, Optimal Graph Kernels [2] etc., we used Random walk graph kernel for our implementation as it offers elegant computation and superior performance compared to the other models available.

For our problem statement, let us consider the original team as T and the team member to be replaced as p and q be the player being considered for replacing p. Let G(T) be the labelled graph for the original team and G(T(p→q)) be the labeled graph for team after replacement. Each graph G is expressed as:

$$G(T):G(\{A,L\}) \qquad \text{------------(1)}$$

where, A: Adjacency Matrix

L: skill indicator Matrix

We used Random Walks to count common walks in the two input graphs G(T) and G(T(p→q)). Now, considering the above graph definition in (1), we have the inputs as:

G(T) :G({A1,L1 })

G(T(p→q)) :G({A2,L2 })

We used Kronecker product of the two graphs to calculate the random walk graph kernel as it gives the same result as individual graph walks. Following equation was used to calculate the similarity score for each prospective replacement player:

$$\boxed{\ker\left(G(T), G\left(T_{p\to q}\right)\right) = y'(Z^{-1} + cZ^{-1}X(I - cYZ^{-1}X)^{-1}YZ^{-1})\left(\left(\sum_{j=1}^{l} L_1{}^{(j)} \otimes L_C^{(j)}\right)x + \left(\sum_{j=1}^{l} (L_1{}^{(j)} \otimes e^{(j)})(I \otimes f^{(j)})\right)x\right)}$$

where, ker(.) = kernel function

$\otimes$ = Kronecker product of two matrices,

c = decay factor,

y and x are the starting and stopping vectors that indicates the weights of different nodes in the graph, where, y = y1 $\otimes$ y2 and x = x1 $\otimes$ x2.

$$Z = I - c\left(\sum_{j=1}^{l} L_1{}^{(j)} \otimes L_C^{(j)}\right)(A_1 \otimes A_c)$$

where, Z^{-1} is precomputed to speed up the calculations as Z is independent of prospective replacement candidate q.

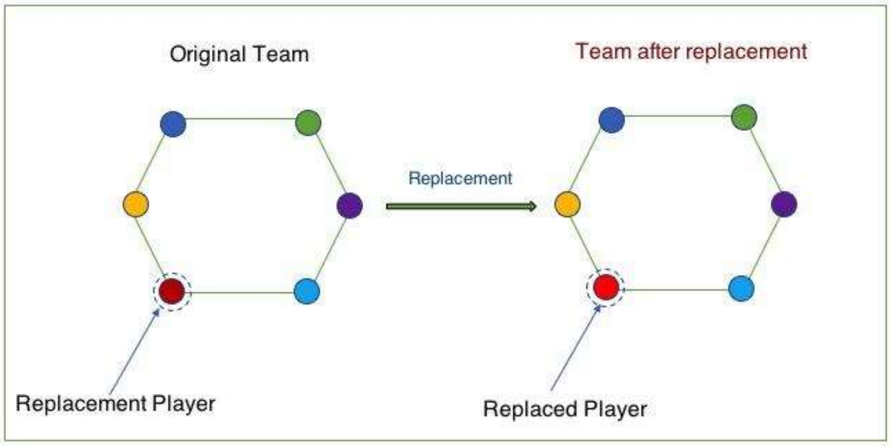


Illustration of the Player Replacement Strategy

Finally, using the similarity scores obtained from the algorithm, team manager can select a candidate for replacing the existing member by picking out the one with highest score.

3. IMPLEMENTATION

3.1. Weekly Dream Team Prediction

In this part, we have explained the three models which we have implemented for the weekly dream team prediction.

3.1.1. Hidden Markov Model

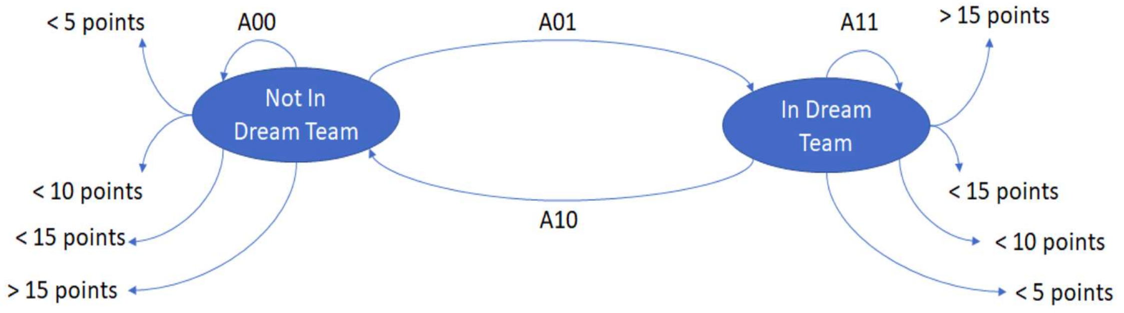
Data Cleaning and Preparation

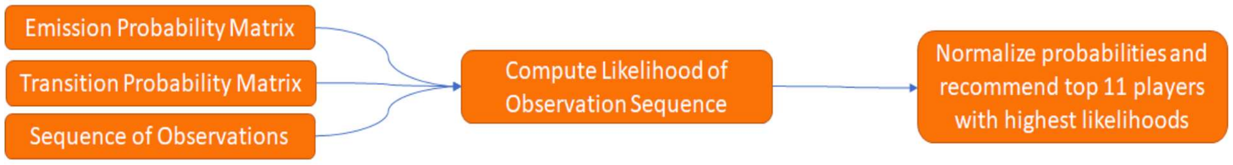
The data set we worked with had multiple features. The dataset had FPL statistics for each player for every week. Majority of the statistics (For example 'goals_scored') in the dataset was aggregated over the weeks up until the week in consideration. So, we cleaned these features to represent the player's statistics for the given week alone instead of the aggregated view. We subtracted the particular week's statistic by the previous week's statistic.

Only for the dream team column, if the player was in the dream team for a particular week, we considered that to be because of his performance in the previous week.

Model Creation

We implemented HMM from scratch and have not used any inbuilt packages. The Hidden Markov Model assumes two things. The probability of a state depends only on the previous state. Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations. Figure shows the HMM for our Dream Team Recommendation task. The two hidden states correspond to 'Not in Dream Team' and 'In Dream Team', and the observations correspond to the total number of points that a professional soccer player scored in a game week.





Emission Probability Matrix
Transition Probability Matrix
Sequence of Observations
Compute Likelihood of Observation Sequence
Normalize probabilities and recommend top 11 players with highest likelihoods

We created a player dictionary, where for every week a player played we stored the total points he scored for that game week and if he was present in next game weeks dream team. We then calculated the probability of a player transitioning from one state to another and the probability that a player emits a given observation from a given state. The sequence of observations in our problem was the sequence of round points that a player scored in every game week up until the game week to be predicted. Thus, the emission probability matrix, the transition probability matrix, sequence of observations and the week to be predicted are the inputs that our model requires. The state in which a player was present after his first game week of the season is his initial state. For every game week up until the game week to be predicted, we calculate the probability of a player being in state 'In Dream Team' or in state 'Not in Dream Team'. For the game week to be predicted, we multiply all the 'In Dream Team' probabilities and the 'Not in Dream Team Probabilities'. We normalize the probabilities and for every player that we predict to be 'In Dream Team', we sort the normalized 'In Dream Team' probabilities in descending order and pick the top 11 players who have the highest normalized 'In Dream Team' probability. The top 11 players that we have picked are our predictions to feature in the given game weeks dream team.

3.1.2. Gaussian Naive Bayes and K Nearest Neighbor

***Files related*:** gaussianNaiveBayes.py, knn.py

Data Cleaning and Preparation

The dataset which we worked on contained the weekly FPL statistics of all the players who played that particular week. Secondly, there were total 82 features given in the dataset out of which the values of majority of the features corresponding to that particular player were the cumulative score of that feature up until that week. While in order to create the model, we need to obtain the data independent of any other week. Therefore, we had two tasks in hand here:

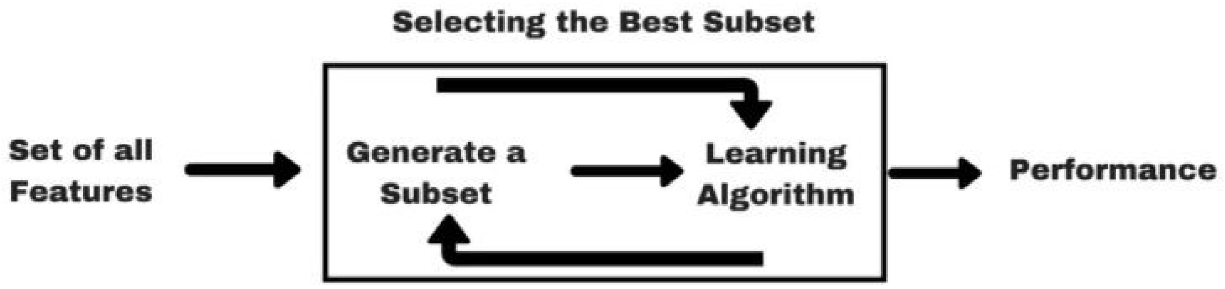
a. Clean the data to remove the dependency from other weeks.
b. Instead of getting the 'per week' data of all the players, we needed 'per player' data of all the weeks.

Therefore, first we observed all the feature values across weeks to understand what all feature values are aggregated and which feature values are independent. Then, for the selected features containing aggregated values, we cleaned the values to represent the player's statistics for the given week alone. We implemented this by subtracting the current week's data values of those selected features from the previous week's data values.

Next, in order to get the 'per player' data, we created a parser that will parse through each individual 'per week' file of all the players and save the data with respect to each player in the data structure 'dictionary of list of list' with the key as the player name. Later, we have implemented feature reduction and model implementation on this data structure.

Feature Selection

In order to select the relevant features, we have used the Backward Elimination Wrapper method. In this method, we start with all the features to train the model. Now, based on the results obtained from the previous model, we remove the features from our subset which are least significant and train the model again. We repeated this step until no improvement was observed on removal of features.



For Gaussian Naive Bayes, the final features we selected to train the model are: PointsLastRound, YellowCards, GoalsConceded, GoalsScored, PenaltiesMissed, Bonus, CleanSheets, PenaltiesSaved, MinutesPlayed, RedCards, BPS.

For K Nearest Neighbor, the final features we selected to train the model are: PointsLastRound, GoalsScored, CleanSheets.

Model Implementation

We have used two models:

a. Gaussian Naive Bayes
b. K Nearest Neighbor

Python Package Used: 'sklearn'

Model	Gaussian Naïve Bayes	Logistic Regression
Package	sklearn.naive_bayes.GaussianNB	sklearn.neighbors

Algorithm used:

a. After the data is cleaned and the features are selected, we will store the final cleaned data into the data structure 'dictionary of list of list' with the key as the player name.
b. We will ask the user to input the week number for which he/she wants to know if the player will be in the dream team for that week or not.
c. For each player,
    1. we will divide his data into train and test where train will contain the data till one week before the current week and test will contain the data of the current week.
    2. Now, we will train the model (Gaussian Naive Bayes or K Nearest Neighbor) for that player.
    3. Then, we will predict weather that player will be in the current week or not using that trained model.
    4. We will store the predicted value of that player in the new data structure.

d. After predicting the output for all the players, we will create the confusion matrix using the predicted value and the actual value and obtain the true positives, true negatives, false positives and false negatives.
e. Now, we will use the true positives, true negatives, false positives and false negatives to obtain sensitivity, specificity, precision and overall accuracy of the trained model.

Also, after observing the trend in data with respect to each player, we discovered that many players were not playing for each week. Therefore, we could not give the value of 'k' in K Nearest Neighbor very large. Secondly, for the initial weeks the amount of data will not be enough as compared to the later weeks. So now let us consider the scenario where the player does not play too many matches in the initial weeks but whatever matches he played he was in the dream team. Now, if we keep the count of k very high and we do not have sufficient training data for that player, the model prediction will be biased towards the class which is in majority. Usually this majority class will be 'Not in Dream Team'. Therefore, to handle these type of cases we decided to take the value of k as 1 and 3. After analyzing the results, k=1 was giving better results than k=3 and therefore, we finally considered k=1 for our data set.

3.2. Weekly Player Point Classification

Data Preparation

The data set we worked with had multiple features. The dataset had FPL statistics for each player for every week. Majority of the statistics (For example 'goals_scored') in the dataset were aggregated over the different week until that week. So, we cleaned these features to represent that player's statistics for the given week alone instead of the aggregated view. This process was pretty simple. We just subtracted the particular week's statistic by the previous week's statistic.

We looked at the points scored('points_scored') by each player every week and created our target variable as 1 for points greater than or equal to 7 and 0 for points less than 7.

***Files related*:** dataPrep.py, PlayerPointClassification.py.

Feature Selection

We did feature selection based on SelectFromModel approach. This approach used to give us the feature importance for all the features we had. Based on certain the feature importance values we chose the top most important features for the model. For this we used python's 'sklearn.feature_selection' package.

For the logistic regression model, we used L1 based feature selection. The packge mentioned above has the implementation for this type of feature selection. Similarly, for the decision tree model, we used tree based feature selection.

Based on these types of feature selection methods, the following were the features we used in 3 of our models (Gaussian Naïve Bayes, Logistic Regression, Decision Tree): 'PointsLastRound', 'GoalsScored', 'CleanSheets', 'Assists'. For SVM, we used the prediction probabilities we get from the previous three models as features.

Files related: dataPrep.py, featureSelection.py

Model Creation

We have used four models.

i. Gaussian Naïve Bayes(GNB)
ii. Logistic Regression(LR)
iii. Decision Tree(DT)
iv. Support Vector Machine as Ensemble of GNB, LR and DT.

Python Package Used: 'sklearn'

Model	Gaussian Naïve Bayes	Logistic Regression	Decision Tree	Support Vector Machine
Package	sklearn.naive_bayes.GaussianNB	sklearn.linear_model.LogisticRegression	sklearn.tree.DecisionTreeClassifier	sklearn.svm

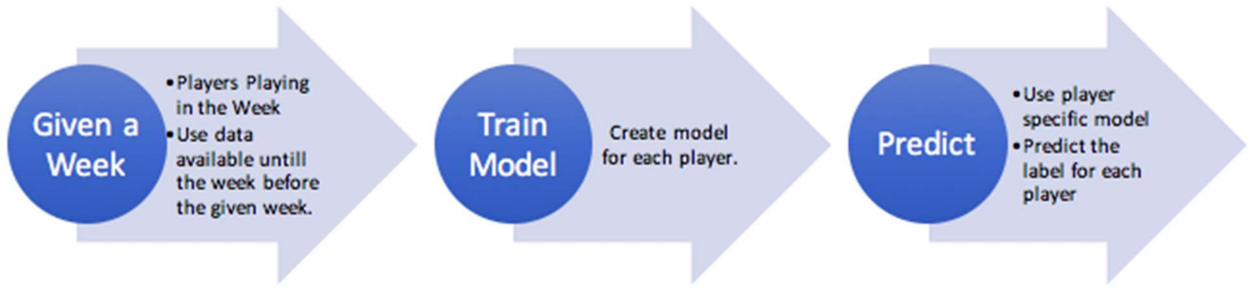
Files related: PlayerPointClassification.py.

Gaussian Naïve Bayes, Logistic Regression, Decision Tree:

These three models take 'PointsLastRound', 'GoalsScored', 'CleanSheets', 'Assists' as feature (FPL Player statistics).

Here we start by considering a particular week 'w'. We then take the subset of the FPL player data till the week 'w' and train the models for every player 'p'. For predicting for a particular player's likelihood of scoring greater than 7 points for a given week, we pick the player specific model and train it with the data till given week and predict for the given week by providing the features for the given week.

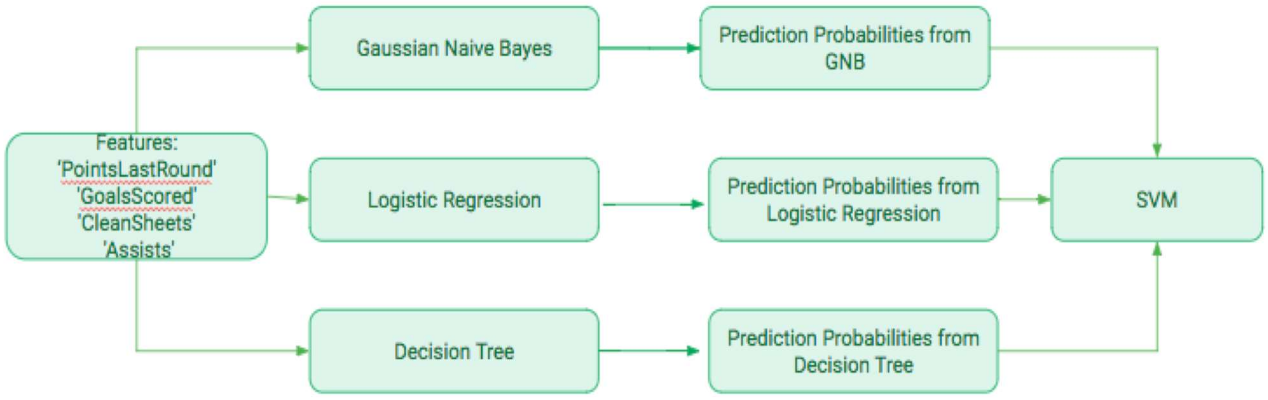
The flow chart diagram below will provide a better understanding of this process:


Given a Week
•Players Playing in the Week
•Use data available untill the week before the given week.
Train Model
Create model for each player.
Predict
•Use player specific model
•Predict the label for each player

Model Flow Chart

Support Vector Machine as Ensemble of GNB, LR and DT

We have used SVM as an ensemble model which ensembles the predictions from GNB, LR and DT models. It takes the prediction probabilities from the previous three models as features.


Features:
'PointsLastRound'
'GoalsScored'
'CleanSheets'
'Assists'
Gaussian Naive Bayes
Logistic Regression
Decision Tree
Prediction Probabilities from GNB
Prediction Probabilities from Logistic Regression
Prediction Probabilities from Decision Tree
SVM

The rest of the modelling procedure remains the same as mentioned in the figure named Model Flow Chart.

3.3. Player Replacement Strategy

Below we will elaborately explain the steps for Player Replacement Implementation.

Data Preparation

Technologies Used: Scikit Learn, Pandas, Numpy and Python.

Steps Followed: To construct the affinity based adjacency matrix and the skill matrix, we needed homogenous data to maintain compliance. Since all the data was from heterogeneous sources, we had to remove outliers, handle messy, inconsistent and un-standardized data. We also combined data from multiple sources and did data scraping of unstructured sources such as websites.

Files related: PopulatePlayerDataFor2014-2015.py

Feature Selection

Technologies Used: Scikit Learn, Pandas, Numpy and Python.

Steps Followed: Feature selection is essential for accurate prediction by the model. It is a subset of relevant features that aid the prediction model.

Features selected for the affinity based adjacency matrix were

- Most Common years that the players have played together.
- Most Common Teams that the players have played together.
- Most Common positions that the players have played together.

Features selected for the Skill matrix were:

- Assists
- Goals
- Red Cards
- Saves
- Minute
- Clean Sheets
- Bonus
- Points
- Yellow Cards

Data Pruning.

***Files related*:** FormatData.py, PopulateDataForEdgeMatrix.py.

Technologies Used: Scikit Learn, Pandas, Numpy and Python.

Steps Followed: As per description, a player would be a good replacement if and only if there exists a connection between the players. This pruning helped us improve the efficiency of decision making and also reduce the complexity by removing sections of the dataset that are irrelevant in making the decision.

Structural matching was performed by assigning scores to the player pool. This score was a combined weight of the three parameters chosen for the affinity based adjacency matrix.

***Files related*:** FormatData.py

Algorithm.

Technologies Used: FormatData.py, PopulateDataForEdgeMatrix.py

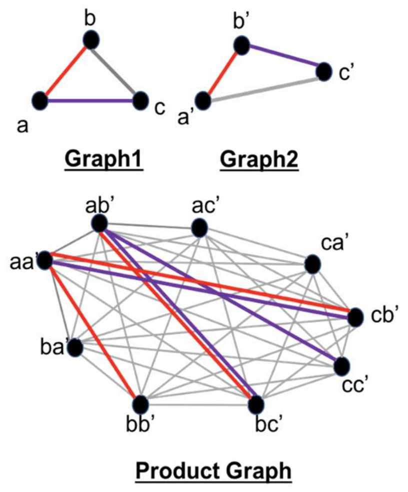
Steps Followed: We have used Fast Random Walk Graph Kernel [1] and a few optimization techniques to find similarity between two players both in terms of the context of the team and also in terms of skill. To aid with finding the similarity between the players in a team setting we utilize the adjacency matrix and to match their skills we use the skills matrix.

Illustration of Random Walk Graph Kernel:

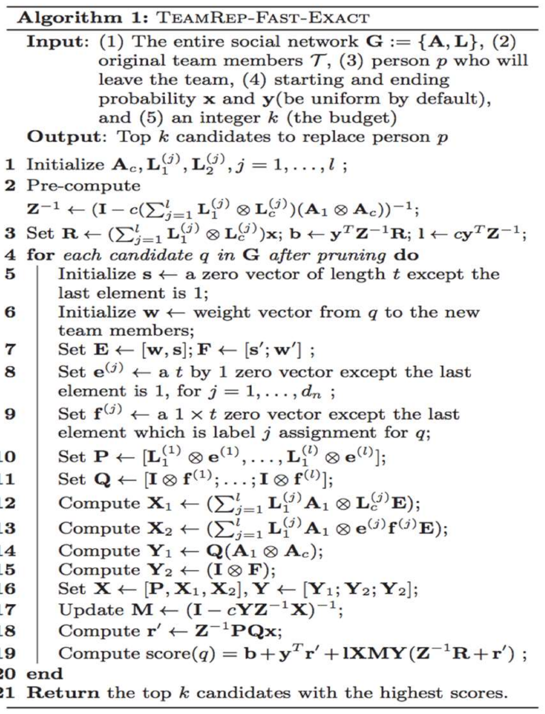
Random Walk Graph Kernel

The Similarity between Graph1 and Graph 2 can be defined as,
"the number of common walks" between the two graphs.

Similarity(Graph1, Graph2) can also be calculated by Random Walk on the product graph.



The following algorithm was implemented to attain the scores which can be used to make the decision on the player can replace an existing player.



```
Algorithm 1: TeamRep-Fast-Exact
Input: (1) The entire social network $\mathbf{G} := \{\mathbf{A}, \mathbf{L}\}$, (2)
       original team members $\mathcal{T}$, (3) person $p$ who will
       leave the team, (4) starting and ending
       probability $\mathbf{x}$ and $\mathbf{y}$(be uniform by default),
       and (5) an integer $k$ (the budget)
Output: Top $k$ candidates to replace person $p$
1  Initialize $\mathbf{A}_c, \mathbf{L}_1^{(j)}, \mathbf{L}_2^{(j)}, j = 1, \ldots, l$ ;
2  Pre-compute
   $\mathbf{Z}^{-1} \leftarrow (\mathbf{I} - c(\sum_{j=1}^{l} \mathbf{L}_1^{(j)} \otimes \mathbf{L}_c^{(j)})(\mathbf{A}_1 \otimes \mathbf{A}_c))^{-1}$;
3  Set $\mathbf{R} \leftarrow (\sum_{j=1}^{l} \mathbf{L}_1^{(j)} \otimes \mathbf{L}_c^{(j)})\mathbf{x}$; $\mathbf{b} \leftarrow \mathbf{y}^T\mathbf{Z}^{-1}\mathbf{R}$; $\mathbf{l} \leftarrow c\mathbf{y}^T\mathbf{Z}^{-1}$;
4  for each candidate $q$ in $\mathbf{G}$ after pruning do
5      Initialize $\mathbf{s} \leftarrow$ a zero vector of length $t$ except the
       last element is 1;
6      Initialize $\mathbf{w} \leftarrow$ weight vector from $q$ to the new
       team members;
7      Set $\mathbf{E} \leftarrow [\mathbf{w}, \mathbf{s}]; \mathbf{F} \leftarrow [\mathbf{s}'; \mathbf{w}']$ ;
8      Set $\mathbf{e}^{(j)} \leftarrow$ a $t$ by 1 zero vector except the last
       element is 1, for $j = 1, \ldots, d_n$ ;
9      Set $\mathbf{f}^{(j)} \leftarrow$ a $1 \times t$ zero vector except the last
       element which is label $j$ assignment for $q$;
10     Set $\mathbf{P} \leftarrow [\mathbf{L}_1^{(1)} \otimes \mathbf{e}^{(1)}, \ldots, \mathbf{L}_1^{(l)} \otimes \mathbf{e}^{(l)}]$;
11     Set $\mathbf{Q} \leftarrow [\mathbf{I} \otimes \mathbf{f}^{(1)}; \ldots; \mathbf{I} \otimes \mathbf{f}^{(l)}]$;
12     Compute $\mathbf{X}_1 \leftarrow (\sum_{j=1}^{l} \mathbf{L}_1^{(j)}\mathbf{A}_1 \otimes \mathbf{L}_c^{(j)}\mathbf{E})$;
13     Compute $\mathbf{X}_2 \leftarrow (\sum_{j=1}^{l} \mathbf{L}_1^{(j)}\mathbf{A}_1 \otimes \mathbf{e}^{(j)}\mathbf{f}^{(j)}\mathbf{E})$;
14     Compute $\mathbf{Y}_1 \leftarrow \mathbf{Q}(\mathbf{A}_1 \otimes \mathbf{A}_c)$;
15     Compute $\mathbf{Y}_2 \leftarrow (\mathbf{I} \otimes \mathbf{F})$;
16     Set $\mathbf{X} \leftarrow [\mathbf{P}, \mathbf{X}_1, \mathbf{X}_2], \mathbf{Y} \leftarrow [\mathbf{Y}_1; \mathbf{Y}_2; \mathbf{Y}_2]$;
17     Update $\mathbf{M} \leftarrow (\mathbf{I} - c\mathbf{Y}\mathbf{Z}^{-1}\mathbf{X})^{-1}$;
18     Compute $\mathbf{r}' \leftarrow \mathbf{Z}^{-1}\mathbf{P}\mathbf{Q}\mathbf{x}$;
19     Compute score$(q) = \mathbf{b} + \mathbf{y}^T\mathbf{r}' + \mathbf{l}\mathbf{X}\mathbf{M}\mathbf{Y}(\mathbf{Z}^{-1}\mathbf{R} + \mathbf{r}')$ ;
20 end
21 Return the top $k$ candidates with the highest scores.
```

Algorithm1: Team Replacement using the Exact Algorithm [1].

Files related: Generate_Graph.m and GraphSimilarity.m

4. RESULTS

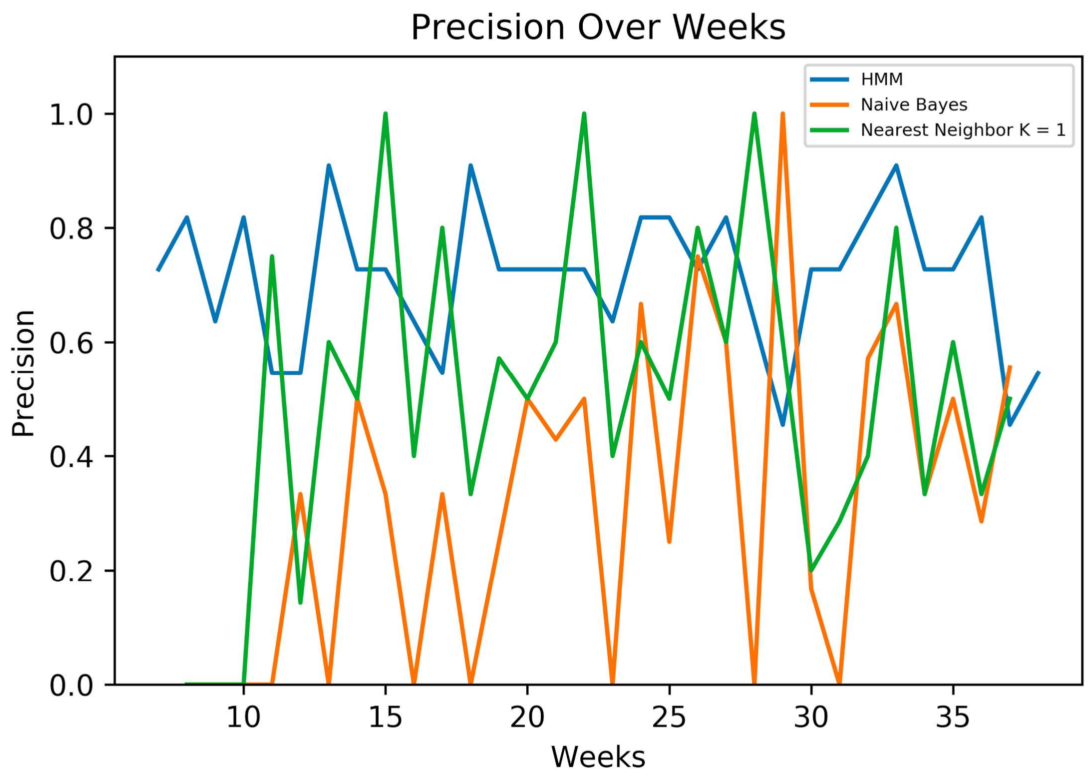
4.1. Weekly Dream Team Prediction Results

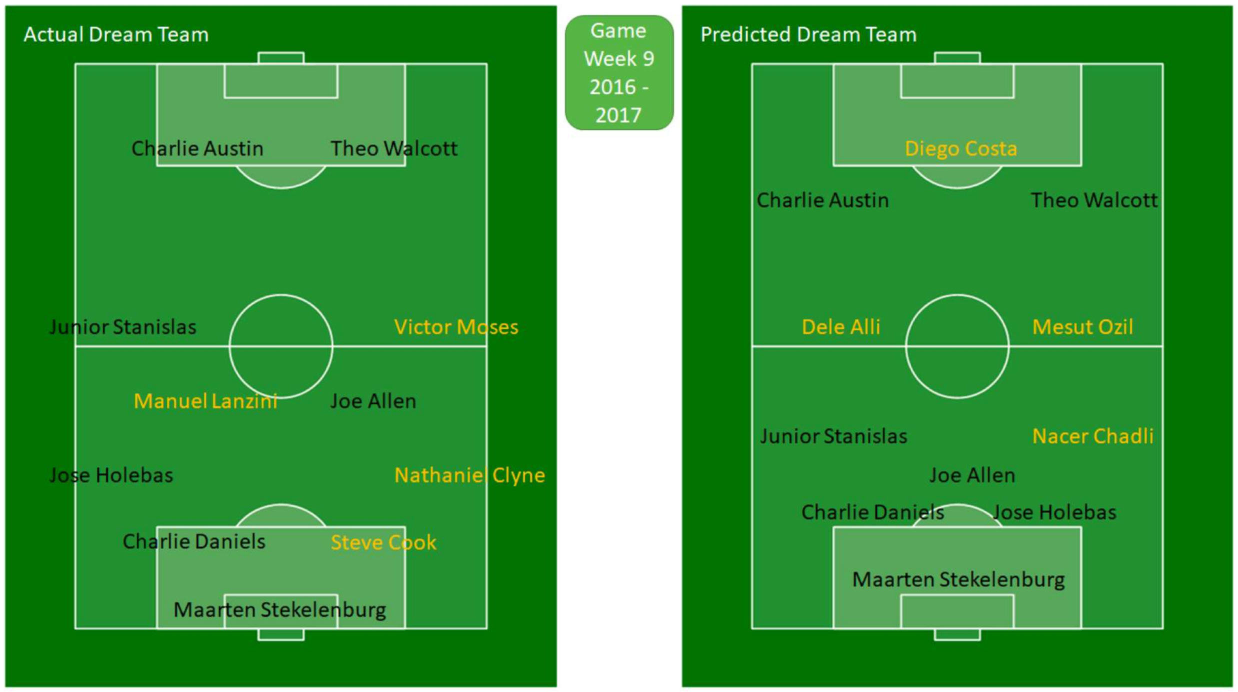
We observed that our Hidden Markov Model incredibly outperformed the Gaussian Naive Bayes and KNN. We attribute this performance to the fact that HMM is a time series model and because of this it is better able to capture the problem scenario.

Number of player in the dream team will always be no more than 11. So our problem deals with choosing 11 players out of close to 500 players. This makes our dataset highly imbalanced. So accuracy could not be considered as the best performance metric. Therefore, we chose Precision as our performance metric. The table below shows the average precisions of different models that we used.

Model	Gaussian Naïve Bayes	KNN	HMM
Average Accuracy	31.7 %	50.5%	71.3%

The graph below shows the performance of the three models over the weeks of FPL 2016/2017 season. HMM consistently gave better precision as compared to the other two models.

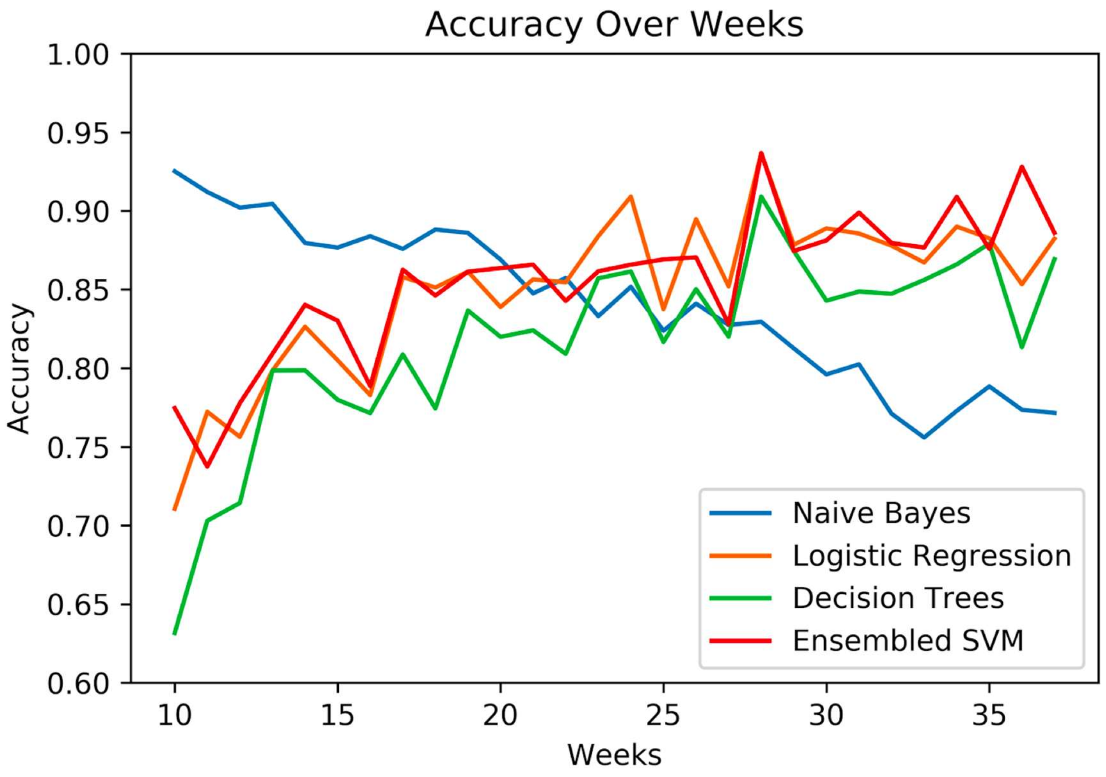




The above figure is a comparison of our Predicted Dream Team vs the Actual Dream Team for game week 9 of the 2016 – 2017 season. The players in black are the ones we correctly predicted and the players in yellow are the ones we wrongly predicted. In the actual dream team, players like Manuel Lanzini and Steve Cook are players who do not play for the top cubs or are superstar players. In place of them, we have predicted Diego Costa, Mesut Ozil and Dele Alli, who are superstar players or play for the big six clubs in the English Premier League. Soccer is a very tricky sport and an underdog team can surprise a star-studded team on its given day. It's difficult to predict all the 11 players to feature in a dream team correctly and to help a FPL user to maximize his points tally, we have another model wherein we try to predict if a given player will score more than 7 points or not in each game week. Scoring more than 7 points in a game week is a good return for a professional soccer player and we try to suggest players who score more than 7 points in a game week, so that a FPL user can stack his team with players who score more than 7 points in a game week and maximize his chances of scoring more overall points for a game week. The dream team is the set of players an FPL user can have, who will give him the maximum return of overall points for that game week.

4.2. Weekly Player Point Classification Results

We observed that the Support Vector Machine as Ensemble of GNB, LR and DT models outperformed the individual Gaussian Naïve Bayes, Logistic Regression and Decision Tree. The accuracy of the ensemble SVM was more and also consistent through several weeks as compared with the other three models as shown in the graph below.



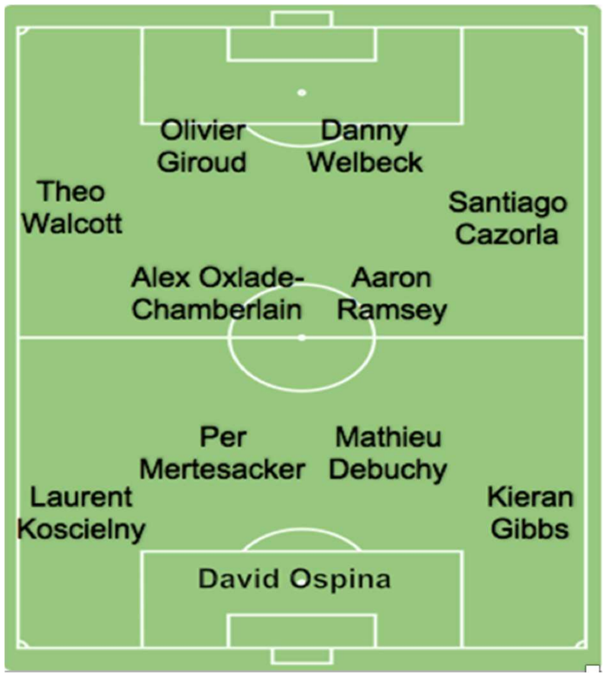
The table below shows the accuracies of the different models averaged over the FPL 2016/2017 season.

Model	Gaussian Naïve Bayes	Logistic Regression	Decision Tree	Support Vector Machine (Ensemble)
Average Accuracy Over The Weeks	83 %	84%	81.1%	85.5%

Despite these good accuracies, we observed that the precision and recall wasn't satisfactory. We wanted to use this model as a pre-processing model which will prune the players to recommend in the Weekly Dream Team Recommendations. But with the unsatisfactory precision and recall values we did not move forward with this plan and kept this section as a stand-alone model.

4.3. Player Replacement Results

As described in the paper [1], we address both Skill Matching and Structural matching, which result in highly accurate predictions.



For illustration of the results, we have considered the replacement strategy for the Arsenal Football Club. Our case study included players from the 2014-15, 2015-16 and the 2016-17 seasons.

As is the case with any football team, the team is categorised as Forwards, Defenders and Midfielders. The test cases were designed in such a way to find replacements for each of the playing positions.

The scores and the most likely player to be replaced were found to be the nearest match.

The image is a map of the original players that were part of the team. The results of replacement are listed in the table below:

	Replacement Player Strategy Results	
Replacement Player	**Potential Replacement Player**	**Score**
Olivier Giroud (FWD)	Joel Campbell (FWD)	5.4867
	Gabriel (MID)	5.2625
Keiran Gibbs (DEF)	Francis Coquelin (DEF)	7.5073
	Gabriel (MID)	7.4819
Alex Oxlade Chamberlain (MID)	Gabriel (MID)	7.74
	Nacho Monreal (MID)	7.7398
Aaron Ramsey (MID)	Gabriel (MID)	7.6372
	Nacho Monreal (MID)	7.6369

Table 1. Results of Player Replacement Strategy

Table 1. shows that the scores of replacement players who play in the same position are higher than the players who play in different positions.

5. CONCLUSION, CONTRIBUTIONS, FUTURE WORK & REFERENCES

5.1. Conclusion

The models for weekly dream team prediction include classification methods and time-series analysis. We can conclude that Hidden Markov Model used for time-series analysis gives us the expected result, while the classification techniques failed to provide us good results due to imbalance in the dataset. Coming to weekly player point classification, we can observe that ensemble methods perform better than the individual models. For our implementation, Ensemble SVM model gives better accuracy than individual models, such as, gaussian naive bayes, logistic regression, and decision tree method. We intended to use this ensemble model to prune our players and use them to make player predictions in the weekly dream team classification. But due to low precision and recall values in the weekly player point classification, we decided to let these two categories independent. Another aspect of the project, player replacement strategy, we correctly identified the best available replacement options for a given player with random walk based graph kernel algorithm.

5.2. Contribution

Common Contributions from all team members
1. Literature Survey 2. Dataset Usage Study 3. Presentation 4. Report Documentation

Table 5.1 Common Contributions

Team Member	Contribution	Percentage
Akshay Jain	Data Preparation, Weekly Point Classification, Player Replacement	16.66%
Ishan Shrivastava	Data Preparation, Weekly Point Classification, Weekly Dream Team Recommendation	16.66%
Kunal Sutradhar	Data Acquisition, Data Preparation, Player Replacement	16.66%
Meha Shah	Data Acquisition, Data Preparation, Weekly Dream Team Recommendation	16.66%
Nikita Shanker	Data Acquisition, Data Preparation, Player Replacement	16.66%
Radha Kannan	Data Preparation, Weekly Point Classification, Weekly Dream Team Recommendation	16.66%

Table 5.2 Contribution Summaries

5.3. Future Work

For the future work, we have identified that the weekly dream team recommendation can be better modeled as a belief-state Markov Decision Process (MDP) [6]. This is because belief-state MDP is able to aptly capture the uncertainty in the player contributions, given the complexity of the domain. Bayesian Q-learning (BQL) algorithm can also prove to be a handy approach to address such uncertainty. Improving parameter selection is another future work to ameliorate the results.

5.4. References

[1] Liangyue Li, Hanghang Tong, Nan Cao, Kate Ehrlich, Yu-Ru Lin, and Norbou Buchler, Replacing the Irreplaceable: Fast Algorithms for Team Member Recommendation

[2] https://www.ethz.ch/content/dam/ethz/special-interest/bsse/borgwardt-lab/documents/slides/CA10_GraphKernels_intro.pdf

[3] Onwuachu Uzochukwu C., P. Enyindah, A Machine Learning Application for Football Players' Selection

[4] Pedro O. S. Vaz De Melo, Virgilio A. F. Almeida, Antonio A. F. Loureiro, and Christos Faloutsos, Forecasting in the NBA and Other Team Sports: Network Effects in Action

[5] Haibin Liu, Mu Qiao, Daniel Greenia, Rama Akkiraju, Stephen Dill, Taiga Nakamura, Yang Song, and Hamid Motahari Nezhad, A Machine Learning Approach to Combining Individual Strength and Team Features for Team Recommendation

[6] Tim Matthews and Sarvapali D. Ramachurn, Georgios Chalkiadakis, Competing with Humans at Fantasy Football: Team Formation in Large Partially-Observable Domains

[7] Fantasy Premier League, https://fantasy.premierleague.com/

[8] Fantasy Overlord, https://fantasyoverlord.com/FPL/history

[9] FPL Archives, http://fplarchives.com/

[10] FPL Analytics, http://fplanalytics.com/history.html

[11] Scikit-Learn: Machine Learning in Python, http://scikit-learn.org/

[12] Using Machine Learning to Predict high-performing Players in Fantasy Premier League, https://medium.com/@277roshan/machine-learning-to-predict-high-performing-players-in-fantasy-premier-league-3c0de546b251

[13] Andrew W. Moore, Hidden Markov Models, http://www.cs.cmu.edu/~./awm/tutorials/hmm14.pdf