Miraaj Chaudhry
Dept. of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854, USA
Miraaj_Chaudhry@student.uml.edu

Football Coach Assistant Agent: Markov Decision Process versus Q-Learning

Abstract—Football is a popular sport in the United States of America as it not only provides entertainment, but provides opportunities for non-sports-related fields to have an impact on the game in professional leagues. One of these fields is data science which has seen an increased role in the sport over the past few years. This role has allowed for advanced analytics and better decision-making on the field. However, despite having their own statistics, teams at the non-professional level aren't able to get as advanced data analytics as the professionals do. This takes learning opportunities for coaches and potentially on-the-field fun for players. This project aims to fix this by using a NFL dataset to create an assistant football coach agent to determine which play is best for a team to run, as well determine possible drives for touchdowns. The data created will also allow for better analysis of past team decisions and actions. In addition to helping amateur teams, the project will allow an increase in fan engagement for both sports and data enthusiasts.

I. INTRODUCTION

Within the United States, American football is the most popular sport as 41% of Americans mentioned it as their favorite sport to watch according to a 2024 survey [2]. As the sport and its professional league (NFL) have gained popularity, so has the use of data science within the sport. Through partnerships with companies such as Amazon's AWS, teams and fans are able to have advanced pieces of data right at their fingertips [4]. These advanced statistics allow fans to feel like a part of the game and understand how their favorite teams and players perform [4]. However, there is a lack of resources widely available to the general public for how this data influences in-game decision-making. Through further research, it was impossible to find an in-game play decider. While professional coaches may not need such an application, die-hard fans would enjoy using it to feel like a part of the game and a member of their favorite team. Furthermore, teams at the amateur and youth levels could use it to help not only win games, but improve their own decision-making skills and learn more about their teams.

Through the use of artificial intelligence and machine learning concepts such as Markov decision process (MDP) and q-learning, the project will analyze the given dataset and determine which play a team should run based on the team's situation (the quarter, current down, and field position). By resolving this issue, the project will not only increase fan engagement in professional leagues, but allow teams at amateur and youth levels to make better decisions which can lead to better experiences for players. It'll allow for a deeper understanding of how successful team decisions are. Moreover, while this exact project is tailored to the football world, it could easily be modified to fit other popular sports, having the same positive impact.

II. PROJECT DESCRIPTION

Before we can cover the different aspects of the projects, it's important to mention the state and actions used for both the MDP and q-learning implementations. In both cases, the states were possible play results in the inputted situation. The user was given an interface to enter their team, the quarter, down mark, and field position. These all represented the current state. The next state would be the result of the play. To simplify, we made the states be 'FirstDown', 'NextDown', 'Touchdown', 'Turnover', and 'NegativePlay'. In both cases, the actions are the play type and formation a team decides to run. Additionally, for both methods we pass a list of rewards. These rewards are based on valuing different play results over others. By default, touchdowns are favored the most, then first downs, then next downs, then negative plays, and finally turnovers. The last two results would have negative rewards.

The project was broken down into several parts to complete it in an organized fashion:

A. DATASET

The dataset being used for this project is 2024 NFL play by play data which was found at **https://nflsavant.com/about.php** on the site called NFLsavant.com. This site was found via research through the "Sports Data Sets" page on The Ohio State University's

College of Arts and Sciences Sports and Society Initiative page. The dataset provides specific details for every play over the 2024 NFL season as well as the play result. A short description of each play is also given.

B. DATA EXPLORATION

Once the dataset was retrieved, we explored the data by using various functions in Python's pandas and numpy libraries. We explored the name of each column and the type of data they each held. We also tried to understand how the data within these columns changed over time while also looking for trends.

We used the data to get a better understanding of how individual teams performed during the season and compared various statistics such as touchdowns and turnovers. The matplotib library was also used in these comparisons to visually see how specific statistics differed from team to team.

We also made sure that the data seemed reliable by making sure there weren't a large number of entries that didn't make sense. For example, when we explored the different play results under each play type and formation, we found that timeouts only occurred under the "UNDER CENTER" formation. After a little more exploration, this fact was confirmed, but due to the quantity of total timeouts, it appeared that the author of the dataset saved them under this category. Due to this and the fact that all timeouts will be ignored, the dataset is deemed reliable.

C. DATA CLEANING

After the data is explored and deemed reliable, the next step is to clean and filter it for our necessary task. The original dataset had a total of 45 columns, most unnecessary for implementing the MDP and Q-learning. Both implementations would only require the user given team's data, but MDP would require only the data in the user given quarter, on the user given down, and in the user given field position. This is mainly due to the fact that MDP calculates values by relying mainly on transitional probabilities from state to state. Q-learning relies on updating q-values after "learning" from new experiences, so all data for the team is needed.

Two functions, markov_reorganize() and q_reorganize() were implemented to complete this job. Both also modified column values and made new columns to make it easy for later functions to determine the current state and the next state. For example, a 'NextDown' column was added to show what the next state would be for a play. A 'IsTurnover' column was also added and it was set to True if the columns for fumbles and interceptions were also both True.

The functions also adjusted the data for special instances such as going for it on fourth down and a play resulting in negative yardage. We compared and modified columns so we were only left with a dataframe that contained the bare minimum of necessary data entries for our problem.

D. MARKOV DECISION PROCESS

Once the data was processed and filtered, the program could begin implementing the Markov decision process to return the most optimal play in the user's situation. An entire class was created for this implementation. After the program intializes this class and sends the given situation, as well as a list of rewards, the class creates two dataframes. One of these dataframes would be for rush plays and one for pass plays. The columns represent possible future states, while the rows are possible actions. The dataframes would save the probability of each action leading to each next state in the given situation. These tables would hold the transitional probabilities, which represent the probabilities of each action, which is also each arrow in the hand-drawn MDP state diagram in figure 1. These probabilities are needed to complete MDP calculations.

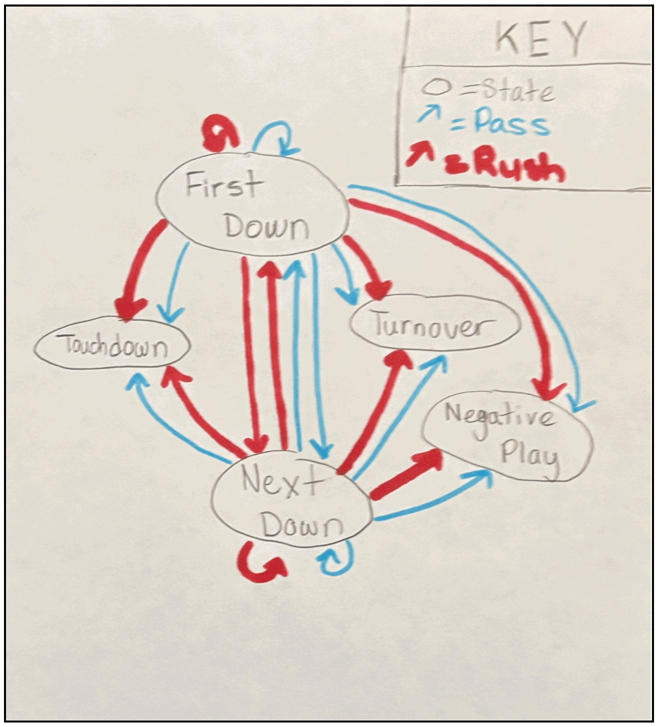

FIGURE 1. MDP state diagram of problem

Once the dataframes were created, for each possible formation, the program read through the data and kept track of how many times each future state occurred as well as how often each formation was called. Once it finished, it simply found the transitional probabilities from the current state to all future states for each possible formation by dividing the two values.

After the transitional probabilities were calculated and saved, the program then used the Bellman equation [1] shown in figure 2, to compute values for each possible action, or play and formation, and then found the max of these values. Since we only care about one play, the value of that last state will be 0, so the discount won't have an impact. The play and formation corresponding to this value was the most optimal play.

Original: $V^*(s) = \max \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$

Adjusted: $V(\text{state}) = \max (\sum_{\text{future states}} (\text{Transitional Probability to next state})[(\text{reward of going from current state to next state})])$

FIGURE 2. Bellman equation: Original adjusted for our problem

E. Q-Learning

The next requirement in this project is to implement the same problem, but by using q-learning. The steps of this implementation started out similar to the MDP. After the necessary data was processed and filtered, the program created a class for this implementation and saved the data and rewards inside this class.

Once the class was intialized, a q-table had to be made to store q-values of possible actions and states. This table would be similar to the dataframes used in the MDP class, but MDP only covered data where the situation matched the one given by the user. Q-learning requires all the data to be read and values to be saved. To accomplish this, the program created five pairs of dataframes. Each pair would represent a sub-table for a quarter (the fifth would be for overtime), while each table inside the pair would represent the field position. The first sub-table in the pair would be for plays when team's are in their own side of the field, the other represented when they're in their opponent's side. A total of 10 sub-tables would be used to represent the necessary q-table. Figure 3 below shows the potential structure of one of these sub-tables.

	Under Center Rush	Shotgun Rush	Under Center Pass	Shotgun Pass
First Down	Q(First Down, Under Center Rush)	Q(First Down, Shotgun Rush)	Q…	Q…
Next Down	Q(Next Down, Under Center Rush)	Q…	Q…	Q…
Touchdown	Q…	Q…	Q…	Q…
Turnover	Q…	Q…	Q…	Q…
Negative Play	Q…	Q…	Q…	Q…

FIGURE 3. Table for one possible quarter and one possible field posiiton such as "1st quarter on Opponent side"

Once the q-table was created, each slot was intialized to 0. It is important to note that rows for touchdowns, turnovers, and negative plays were created, but these q-values always remain at 0 as q-learning only updates the current state's q-values and no plays can run at these three. They were still included to ensure all the necessary details of the problem are involved and to make calculating the q-values easier.

The program had to then fill the table, but this time the q-learning formula [7], shown in figure 4, would be used to update the q-value corresponding to the current state and action taken in data entry.

Original: $Q(s, a) \leftarrow Q(s, a) + (\alpha)[R(s, a, s') + (\gamma)(\max(Q(s', a')) - Q(s, a)]$

Adjusted: $Q(\text{current state, formation}) \leftarrow Q(\text{current state, formation}) + (\alpha)[(\text{reward of going from current state to next state via formation}) + (\text{discount})(\max(Q(\text{next state, formation}) - Q(\text{current state, formation}))]$

FIGURE 4. Q-learning formula: Original adjusted for our problem

Once all the data was read, row by row like it was for the MDP, the program looks at the user's inputted situation and locates the sub-table corresponding to it. Then inside that sub-table, it finds the row with the current state and finds the formation with the max q-value. For example, if the user inputted scenario was 1st quarter, 3rd down, on the opponent side of the field, the program would go to the first pair of sub-tables, go to the table with opponent fild position plays, and look for the row 'NextDown'. Once it locates this row, it'll look for the max q-value in it and note the formation. The formation with the max q-value is the optimal play.

F. Hidden Markov Model

Even though it wasn't originally planned, a hidden markov model was added to the agent to allow it to generate sequences of play results that are probable to end in a touchdown. Essentially, it is outputting drives that will end in a score. To accomplish this, the program identifies hidden states as pass plays and rush plays. Formations were ignored for this model, as time didn't permit them to be considered. The observed states are the states from the MDP and q-learning: first down, next down, touchdown, turnover, and negative play.

Like the previous two sections, the program begins by initializing a hidden markov model class. This class uses the same filtered data as the MDP, focuses on making a dataframe on transitional probabilities between each hidden state, and then a dataframe on transition probabilities from each hidden state to each observed state. The program uses the same iterative method as it did in the MDP class to calculate these probabilities.

It first calculated the transition probability of each hidden state going to the other or itself. Then calculated the transition probability of going from each hidden state to an observed state.

Once this data is saved, the program calls a function to generate a sequence of play results until either 15 plays have been generated, which is a good upper limit on plays in drives, or until a touchdown is produced. The program generates these sequences by first randomly selecting a hidden state, assuming each has a probability of 0.5 of happening at the start of a drive. It then randomly chooses an observed state by using the probabilities calculated earlier with the function np.random.choice(). Then another hidden state is randomly chosen based on the probabilities of hidden states transitioning between one another or itself, and then another observed state will be randomly chosen. This process repeats until the 15 play limit is reached or a touchdown occurs. If the function outputs the same sequence multiple times for a situation (as it did in an example in the project notebook), then a touchdown is less likely in that situation. The program also ensures that the play results can only result in situations that are possible by making sure that it cannot generate any down other than 1 after a fourth down.

III. EXPERIMENT EVALUATION & RESULTS

All three implementations output data that is correct. When the MDP and q-learning methods were being programmed, their tables and calculations were outputted to show how the final answer was found (the showing of tables and calculations were removed for the final version). Based on this, the final results seem accurate based on the formulas used.

The MDP returns the optimal play in the form of a list where the first element is the play type and play result, the second is the formation, and the third is the value calculated through Bellman equation. Q-learning also returns its decision in a list where the first element is the formation with play type and the second is its q-value in the given state. Let's further evaluate the results of this program.

A. Q-LEARNING

To further evaluate the results of the q-learning implementation, we took several steps. The first of which was making a series of different reward sets. These sets varied in valuing different play results. For example, some valued touchdowns more than anything else, some valued first downs, some saw first downs equal to touchdowns, etc. We also included a set that valued turnovers and negative plays over first downs and touchdowns. We tested the same scenarios with these varying reward sets and provided a table with them in the project's notebook. When comparing the entries, it was important to compare different reward results of the same team and not compare results of different teams together (this will be done later). Based on the table, altering the reward set most of the time had no effect on the suggested play, as long as the actions we valued stayed the same. The one reward set that was widely different was the one that valued negative plays and turnovers. The other reward sets, while having different values, were preferring the team to score or pick up a first down, while this set was hoping they'd make a mistake.

Essentially, this mistake-seeking reward set showed us the worst play a team can run in a given scenario. It develops the idea that altering the rewards for a task in q-learning, allows us to learn different pieces of information from the data. We developed this idea further by using a couple more reward sets. One set all the rewards to 1, essentially trying to see if we can see the frequency of each action occurring in a given state. We also made a reward set with touchdowns and first downs having equal non-zero values (10), but all other rewards are 0. This set was used to get a rough idea of which team was more successful in running certain plays in certain situations.

All this information can easily be obtained by iterating through the dataset itself, but this analysis was done to see how altering rewards affects q-learning decisions and q-values, but more importantly to show that altering them can provide us with more information on the data.

B. MDP vs Q-LEARNING

Now it was time to compare the MDP results to the q-learning results. To do this, 10 random scenarios were made and passed to each of the methods. After we received each of their suggested plays for each scenario, we had to determine which method was actually finding the "optimal" play.

Just by analyzing the results, it was easy to see that results vary. Some returned the same play type (run or pass) but different formations, or vice versa. Some returned completely different plays, but none were seen to return the same exact plays. The values associated with the methods shared the same variance, so more analysis was required.

We iterated through the filtered dataset, the one used for markov models as it had the necessary data, and used it to find the average reward of calling each method's play and formation in the given scenario. We ran this a few times to get a sense of the trend and found that the average reward for both methods stayed within 1.0 of one another. In fact, it was within 0.5 for most tests. In some cases the MDP had a higher reward, in others q-learning had a higher reward.

Due to this closeness and varying results, we ran one more calculation and found how many times the MDP suggested play was called in the given situation versus the q-learning suggested play. The q-learning play had a much higher frequency of being called which makes sense as q-values are always being updated. We could make this comparison as both result in similar average rewards, but q-learning doing it with a much higher sample size is far more impressive. In the final comparison seen in the project's notebook, on average, the number of q-learning suggested plays actually being called in almost 5 times as high as the MDP ones!

C. Hidden Markov Model

The results of the hidden markov model are not explored in-deptly in the current project as it was not the main focus. However, based on the dataframes that the class creates, it appears the generated sequence are plausible. The results of this method can definitely be used along with either MDP or q-learning.

The results of the hidden markov model are in play results and not plays, but these results can be used with the tables made by q-learning and the MDP to decide which plays to run.

IV. LIMITATIONS

The project overall does have some limitations that should be addressed. It is important to remember that this is meant to be a simple agent that can decide plays and help with analyzing teams, but it is targeted towards amateur, recreational, and youth athletes. Due to this, this program should not be expected to complete the task in the same manner as it would in a professional team or league.

The program doesn't take all possible factors into consideration when determining the most optimal play. Realistically, the program should also consider how many yards a team needs to pick up a first down, the opponent and their strengths, timeouts, end of quarters, and many more factors that will make the MDP and q-learning equations much more complex. Due to this, this project was designed to complete this complex task in a simplified way.

Another limitation is that the program assumes that the team has already decided to run a play on offense. It shouldn't be used to determine if a team should run a play in a certain situation, but what play to run if they've already decided to run a play. The hidden markov model also assumes that the probability of pass and rush plays being at the start of drives is 0.50 to 0.50. This is mainly because the data would need to be reprocessed and modified to determine the beginning and end of drives, and time did not permit this.

V. ONLINE SOURCES

A few sites were visited to help in programming this project. The main one was the official pandas library website [6] which was used to find and understand a variety of pandas functions. The numpy [5] and matplotlib [3] official websites were also used to understand the few functions used from these libraries.

The Geeks for Geeks official website was also used to understand the concepts of the Markov decision process [1] and q-learning [7]. Professor Alam's Artificial Intelligence lectures numbers 5 and 6 were used for the same reason.

ChatGPT was consulted with these websites to gain a better understanding of concepts and find possible functions to complete a task. ChatGPT was NOT used to create large segments of code for this project as the code is original.

VI. CONCLUSION

From this project we can conclude that artificial intelligence and machine learning concepts can be used effectively for all in the sports world. They can complete a variety of tasks from assisting in play decision-making to aiding in analyzing team performance.

The main goal of this project was to compare results of the Markov decision process to q-learning. From the project, it's clearer that if a large dataset, like the one used, is available, q-learning is the more effective method in making optimal decisions. However, if there's a smaller amount of data, then the Markov decision process is likely a better option. In terms of football and sports, the MDP is likely the best option early in the season, but eventually q-learning will become more effective as the season goes on.

REFERENCES

1. "Bellman Equation." *GeeksforGeeks*, GeeksforGeeks, 24 Feb. 2025, www.geeksforgeeks.org/bellman-equation/.
2. Jones, Jeffrey M. "Football Retains Dominant Position as Favorite U.S. Sport." *Gallup*, Gallup, 26 Feb. 2025, next.gallup.com/poll/610046/football-retains-dominant-position-favorite-sport.aspx
3. *matplotlib*, https://matplotlib.org/
4. "NFL and Amazon Web Services Expand Partnership to Further Shape the Future of Football: NFL Football Operations." *NFL Football Operations*, operations.nfl.com/updates/football-ops/nf;-and-amazon-web-services-expand-partnership-to-further-shape-the-future-of-football
5. *NumPy*, numpy.org/.
6. *pandas*, pandas.pydata.org/.
7. "Q-Learning in Reinforcement Learning." *GeeksforGeeks,* GeeksforGeeks, 25 Feb. 2025, www.geeksforgeeks.org/q-learning-in-python/.