CS536 Final: Data Completion and Interpolation

Student Name: Xuenan Wang

NetID: xw336

1. The Dataset

I selected a database for European soccer. This is an open-source database on Kaggle.com(https://www.kaggle.com/hugomathien/soccer) shared by Hugo Mathien. I slightly reformatted some data and uploaded it into my online MySQL database which I set open and will be kept open for a month (2020.12) unless being hacked. Connection detail in appendix 2: source code.

The database is consisting of 4 tables (Country, League, Team, Team_Attributes). Detailed table structure is showed in appendix 1: database structure.

This database provides detailed team information for hundreds of European soccer teams. I'm going to analyze the teams from five major leagues which represent the top level of European soccer between 2010-2015. For same league teams in the same year, I can train a model to learn the relationship between their playing genres and season final ranking. This model will indicate which playing genre combination is relatively successful for this league in this season. Some thoughts and reasons behind this idea are as following:

1. As time goes, the leading playing genre changes. But this kind of genre changing took a few years to happen and if I compare two consecutive years or even three, it's possible we can see this change. Since these data are all from previous years and we already know all about the soccer analyzation in these years, so we can easily verify our results.
2. The playing genre between different leagues are different. For each league in five major leagues, they all have distinct and lasting genres. We can compare a league genre with another one and see the results.
3. Top teams in each league will play UEFA Champions League, an annual club soccer competition to pick the best European soccer club of the year. The genre of the champion club and final round clubs are thought as the best performed playing genre of the year. This genre should be somehow related to best ranking genre of each leagues.

2. Define the Problem

We can take data of team playing genre from one league in one year as a basic data set. I visualized it as following:

$$\begin{bmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{bmatrix} \Longrightarrow \begin{matrix} Y_1 \\ \vdots \\ Y_m \end{matrix}$$

For each team, there are n features, and all these features will form a playing genre. There are m teams in one league. Y_m represents the final league ranking of team m.

For each feature in a team, there are two measurements: digit and class. To pick one, we need to understand one thing first. The digit is related to the ability of this team which for modern soccer world is related in the financial strength of a club. And a champion club is much more lucrative than a low-ranking club. That's going to cause a problem that all high-ranking clubs will have relative higher digit than low-ranking clubs, which indicates that the digitized features are biased by actual world club financial strength. Since I'm trying to analyze the relationship between playing genre and ranking, it's better I choose the classified measurements which is unbiased by financial situation. The classified measurements, as name indicates, are discrete data. Therefore, I'm going to deal with discreate data for future model training and predicting.

To simplify my life and also make this more real-life meaningful, I also made ranking data Y as discreate data as following rules:

$$TOP: Y_1 \cdots Y_c$$

$$MIDDLE: Y_{c+1} \cdots Y_d$$

$$TAIL: Y_{d+1} \cdots Y_m$$

I classified Y into 3 groups. This classification has real-life meaning.

Take Spanish top level soccer league (La Liga) for example.

a) Top 4 clubs (from previous year) will participate in UEFA Champion League, 5th and 6th (from previous year) will participate in UEFA Europa League. Both of these two competitions are high-level soccer matches between top-level clubs in Europe. These two competitions will take place at the same time as domestic league going on. For clubs taking these competitions, they have to be very good to take both domestic league and UEFA league. The fact that these clubs can participate UEFA leagues implies they are above other clubs and that's the top level for every domestic league.

b) For 7th to 17th ranking clubs, they are the middle part for every league, and they represent the average level for each league.
c) For 18th to 20th ranking, it's the downgrade zone. It means that after this season, these 3 clubs will be downgraded to lower-level league because of their poor performance. The fact that they will be downgraded indicates they cannot represent the level of this league, and they had a terrible performance for this season and presumably, their playing genre are not so good as for ranking.

It's clear that by classifying Y, we have a more real-life scenario for our problem. From there, we have the problem defined as following:

Problem: Take club genre data from one league in one year, train a model, so that for a certain club genre data we can predict the ranking status as in TOP/MIDDLE/TAIL format I defined earlier.

3. Model

Based on my previous analysis on dataset, the model needs to deal with multi-classification with discrete data.

Notice that for each model I train, I only have $m \times n$ size of data and typically, $m = 20$ and $n = 11$. This means for each model the training data size is really limited. Recall as I explained, combining different leagues and different years is meaningless because there are different playing genres between different leagues and playing genre in same league will change over time.

To sum up, I'm facing a multi-classification problem with limited size of discrete data for training.

My selection of machine learning model is **decision tree**.

Decision tree is simple but powerful in this situation. To train a decision tree, I don't need a lot of data. And predicting using decision tree is not time-consuming and do not require heavy computation. Also, decision tree is designed for discrete data. It meets all my needs here.

Input data for this model will be a $m \times n$ matrix X, where X_{mn} represents the data for attribute n for team m, and an array of discrete data of ranking abstraction, which will contain three different kind of data. Output data will be a decision tree class I defined. In this class, there will be functions to print out the tree structure, predict result of a new data, and calculate the errors.

To evaluate my model, I can calculate the error using the exact same training data for this model. Since there will be situations for exact same X having different Y, it's reasonable I introduce probability for some branches. To understand this, consider the following example:

Levante UD: ['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']

RCD Mallorca: ['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']

Above are two team playing genre from 2012 La Liga. Levante UD ranked 6th and RCD Mallorca ranked 8th, which means they have exact same playing genre but labeled as "TOP" and "MIDDLE". This is reasonable because playing genre doesn't guarantee anything. The final ranking is also affected by player skills and their abilities to carry out the strategy. My task here, however, is to find the relationship between ranking and playing genre. So when facing this situation, it's reasonable to say that both situation could happen, and to decide which is the case, I pick them randomly following the probability distribution.

By introducing the probability, we are facing situations where the training data won't have exact outcome we exacted. I define the accuracy of the model as following:

$$ACCURACY\ OF\ MODEL = \frac{\sum_{i=0}^{m} \delta(X_i)}{m}, where\ \delta(X) = \begin{cases} 0\ if\ real(X) \neq predict(X) \\ 1\ if\ real(X) = predict(X) \end{cases}$$

Ideally, if there is no need to introduce probability into my model, the accuracy should be 1. The lower this accuracy is, the more non-distinct original data is, and the model will product worse results.

For this problem, I don't really have irrelevant data when I'm training. It's decision tree's responsibility to exclude those irrelevant attributes and they won't be a problem.

As far as I can see, the data is complete. I need to manually remove some data in order to product data completion experiments. Since the data is manually removed, it's easy to compare restored data with original data. The detailed experimental design is in the 4th chapter: Experiment Design.

Now that I'm clear about the model, I need to think about the training algorithm.

The training algorithm I choose is ID3, but I changed a little bit over it. I explained above that there are situations where I have to introduce probability into training. To be exact, when there are multiple X having the same values, I predict Y based on $Y_{training}$ probability distribution.

I call this modified algorithm **probability-ID3**. Detailed mathematical description are as following:

$$H(Y) = -\sum_{y} \mathrm{P}(Y = y) log \mathrm{P}(Y = y)$$

$$H(Y|X = x) = -\sum_{y} \mathrm{P}(Y = y|X = x) log \mathrm{P}(Y = y|X = x)$$

$$H(Y|X) = -\sum_{y} \mathrm{P}(X = x) H(Y|X = x)$$

$$IG(X) = H(Y) - H(Y|X)$$

The Probability-ID3 Algorithm:

- **For each variable X_i, computer $IG(X_i)$**
- **Select the variable with maximum information gain**
  - **If the maximum information gain is zero, then calculate the probability distribution of Y**
  - **Store the probability distribution in node, return**
- **Split/Partition the data based on that variable**
- **Recursively apply this algorithm on each part of the data**

---

4. Experiment Design

A. Evaluation of Model

For a model D, I train it using data X_{train} and calculate the accuracy of that model. The accuracy should be different every time it calculates unless it equals to 1. Train the model with 6 different data X_{train} and calculate the accuracy for 50 times. Plot the outcome as histogram.

B. Ranking Prediction

Train a model for certain league and certain year T, take the club data from year $T+1$, predict the ranking. Compare the outcome with actual team ranking, analyze the result.

C. Comparison of Different League

Train models for different league at same year, print out the model and analyze the result. Compare the result with real-life situation.

D. Comparison of Different Year

For a certain league, train models for different year. Print out the model and analyze the result. Compare the result with real-life situation.

E. Further Analysis of Prediction Accuracy

Redo experiment B and for this time, focus more on different years and leagues. Analysis the result.

5. Results and Analysis

A. Evaluation of Model

I trained a model using 2012 La Liga data, results are as following:

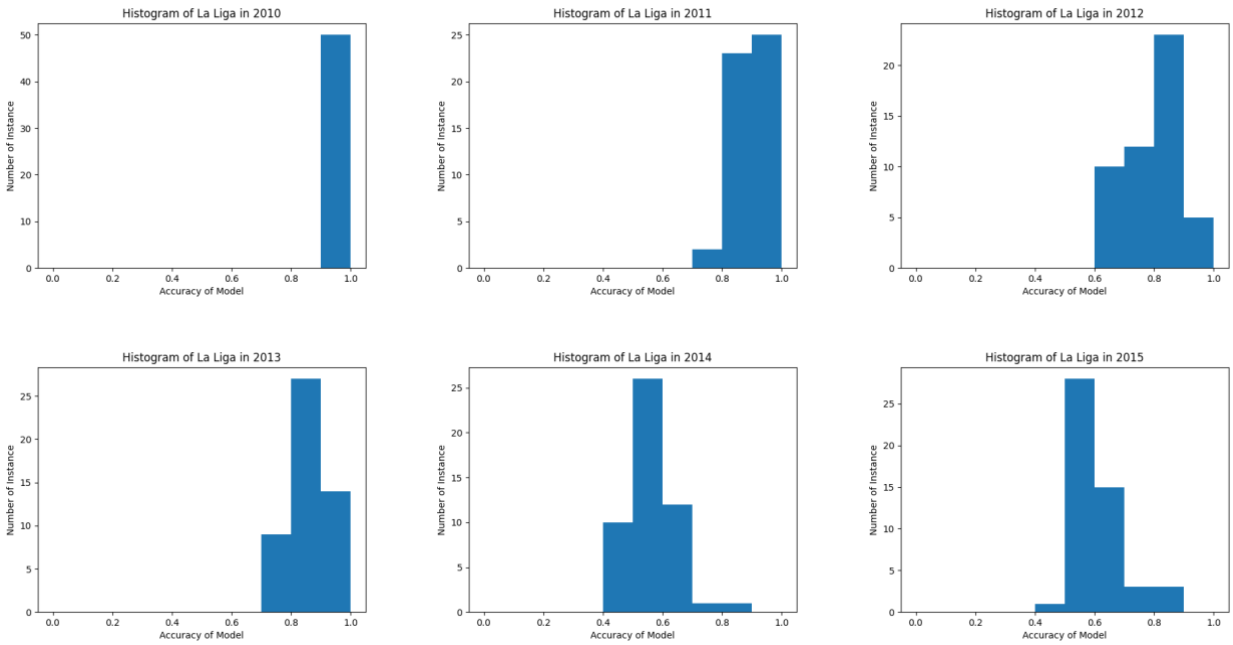
```
xw336@ilab3:~/MachineLearning$ /usr/bin/python3 /ilab/users/xw336/MachineLearning/Final.py
X:                                                                                                                                                  Y:
['Balanced', 'Little', 'Mixed', 'Free Form', 'Risky', 'Normal', 'Lots', 'Free Form', 'Medium', 'Press', 'Wide']                                      TOP
['Slow', 'Little', 'Short', 'Free Form', 'Normal', 'Little', 'Normal', 'Free Form', 'Medium', 'Press', 'Normal']                                     TOP
['Slow', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                     TOP
['Balanced', 'Little', 'Mixed', 'Organised', 'Risky', 'Lots', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                    TOP
['Balanced', 'Little', 'Mixed', 'Organised', 'Risky', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                  TOP
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                TOP
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Wide']                                  MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Lots', 'Normal', 'Organised', 'Medium', 'Press', 'Wide']                                    MIDDLE
['Slow', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                    MIDDLE
['Slow', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Wide']                                      MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Short', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                               MIDDLE
['Balanced', 'Little', 'Mixed', 'Organised', 'Risky', 'Lots', 'Normal', 'Organised', 'Medium', 'Press', 'Wide']                                     MIDDLE
['Slow', 'Little', 'Short', 'Organised', 'Normal', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                    TAIL
['Balanced', 'Little', 'Mixed', 'Organised', 'Risky', 'Normal', 'Normal', 'Organised', 'Medium', 'Press', 'Normal']                                 TAIL
['Balanced', 'Little', 'Mixed', 'Organised', 'Normal', 'Lots', 'Normal', 'Organised', 'Medium', 'Press', 'Wide']                                    TAIL
```

Above is the original data, my decision tree is as following:

```
--- New Level ---
feature #: 3
result: {'Free Form': 'TOP'}
--- New Level ---
feature #: 10
--- New Level ---
feature #: 4
feature #: 5
result: {'Normal': 'MIDDLE'}
--- New Level ---
feature #: 2
feature #: 5
result: {'Lots': 'TOP'}
feature #: 4
result: {'Risky': 'MIDDLE'}
--- New Level ---
feature #: 0
feature #: 0
result: {'Balanced': 'MIDDLE', 'Slow': 'TAIL'}
feature #: Prob
prob: [[0.5, 'TOP'], [1.0, 'TAIL']]
feature #: Prob
prob: [[0.5, 'MIDDLE'], [1.0, 'TAIL']]
--- New Level ---
feature #: Prob
prob: [[0.5, 'TOP'], [1.0, 'MIDDLE']]
feature #: Prob
prob: [[0.16666666666666666, 'TOP'], [1.0, 'MIDDLE']]
```

We can see that for La Liga in 2012, the most influential attribute is #3 (Play Positioning), and there are some attributes that is not very deterministic for the result, hence probabilities present.

To show the accuracy of my model, I trained La Liga data from 2010-2015 and calculated the accuracy of model for 50 times. Following are my results:



We can see that the accuracy is very good for 2010 and 2011, adequate for 2012 and 2013. However, for 2014 and 2015, it's not very good. The accuracy of my model is decreasing as time goes, it's an interesting result. This imply that as time goes, the playing genre is not as much difference as it used to be. A possible explaination is that teams will try to learn from the top ranking teams to improve their performance. As they compete each other, they learn and improve. The fact that accuracy is changing over time is a proof for that.

However, to verify my explaination, I need to see the decision tree structure. Hence following experments.

```
xw336@ilab3:~/MachineLearning$ /usr/bin/python3 /ilab/users/xw336/MachineLearning/Final.py
Measurement by Level for 2010: [[5], [7], [8, 3], [9, 10, 9, 0], [4]]
Measurement by Level for 2011: [[3], [9], [7, 0], [5, 10], ['Prob', 4], [5], ['Prob']]
Measurement by Level for 2012: [[3], [10], [4, 5], [2, 5, 4], [0, 0, 'Prob', 'Prob'], ['Prob', 'Prob']]
Measurement by Level for 2013: [[2], [3, 4], [10, 0], [10, 5], ['Prob', 3], [6], [7], [10], ['Prob']]
Measurement by Level for 2014: [[7], [4], ['Prob', 5], ['Prob', 0], [2], ['Prob']]
Measurement by Level for 2015: [[3], [7], [0], [4], ['Prob']]
```

Above is the decision tree node for each level. We can see there are certain attributes are more deterministic than others. For example, #3 and #7 are more deterministic than others because they show up at higher levels.

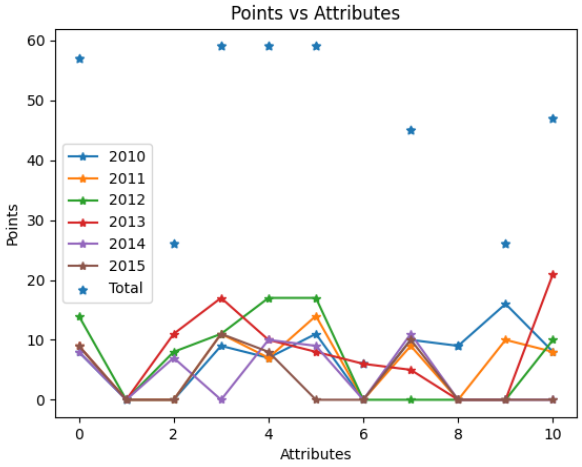
Above result is telling us things, but not clear enough to understand it. To make it clear, I define a weighted points. Defination is as following:

Since there are total 11 attributes, there will be at most 11 levels for my decision tree. For atrribute $X_i, i \in [1, 11]$

$$POINT(i) = \sum_{j=1}^{m} w(i)$$

$$w(i) = 12 - l, l\ is\ the\ level\ number\ of\ i$$

I calculated the points based on above definition. Following is the result:



We can see that #0, #3, #4 and #5 are more significant comparing to others.

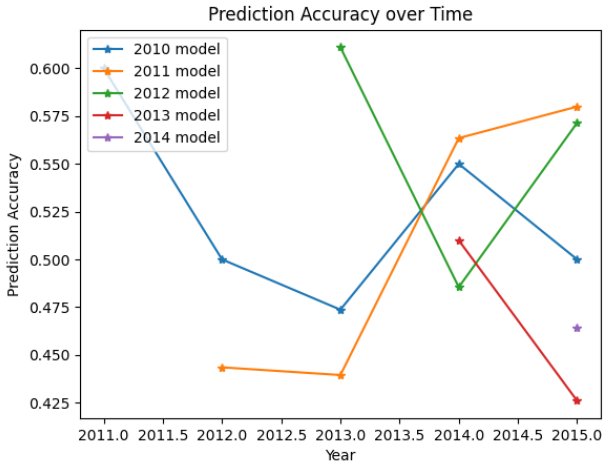
To understand this result, we need to know what these attributes are. #0 is "Play Speed", #3 is "Play Positioning", #4 is "Creation Passing", #5 is "Creation Crossing". These really make sense because La Liga is a league focus on teamwork, positioning and creation. My result accurately captured this. Notice that #1 have a point of 0, which means it have no contribution to classifying ranking. If we look at the data, #1 is "Play Dribbling" and all La Liga teams choose "Little" for this one. This is an attribute reflect how much this team rely on single player. As I mentioned, La Liga focus on teamwork, so it makes sense no team encourage single player dribbling.

Up to now, my model has an acceptable accuracy and can capture certain information for a league. This is a very good beginning and I'm sure there will be more underlying information as experiments goes on.

B. Ranking Prediction

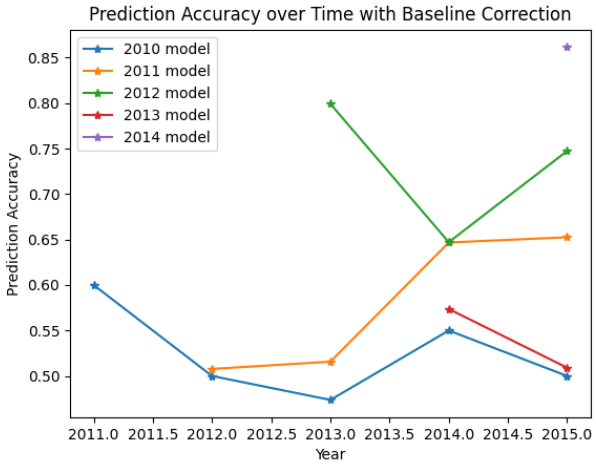
To predict a ranking, I need to train a model and take data from same league in the next few years to test and predict that ranking. Remember I introduced probability into my model, therefore the actual success rate won't be very high considering the baseline, which is the accuracy of model discussed in previous problem. Also, to avoid the influence of probability, I need to repeat every prediction for same

data on same model for multiple times and take the average successful rate. For the following experiment, I repeated each prediction for 100 times.



Above figure might be confusing. Let me explain. The Y axis is the prediction accuracy. It's calculated by averaging the prediction success rate for the same test data on same model over 100 times. The X axis is the year. For model trained by 2010 data, the prediction rate will be calculated on [2011, 2015]. You might notice that the X axis starts with 2011 because I won't be predicting 2010 data since I don't have data before 2010. Following the same logic, for model trained by 2011 data, the prediction starts with 2012, and so on.

Ideally, the prediction accuracy should be gradually decreasing over time, because the playing genre will be less and less related over time from previous seasons. However, my result is not saying the same thing. There are some decreasing patterns, but also significant increasing patterns exist. Aside from this, the prediction accuracy is too low to be meaningful. Notice that with accuracy around 40% for a three-class prediction, it's not significantly improved comparing to random pick. However, there's the influence of probability. Therefore, I plot the below figure to counteract the probability influence.



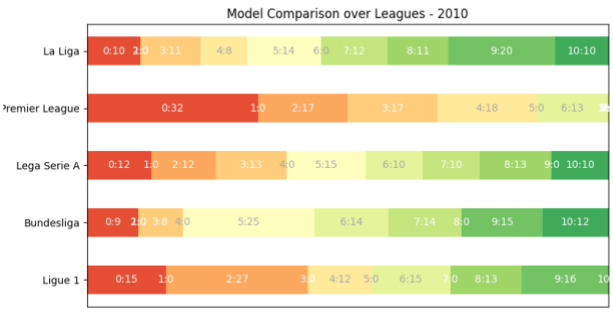
Before predicting results using testing data, I calculated a baseline using training data on trained model for 100 times and take the mean. When plotting the prediction accuracy, I take the accuracy over baseline so that the accuracy will be offset of probability influence. We can see that without probability influence, the prediction accuracy is better although not good enough for actual prediction.

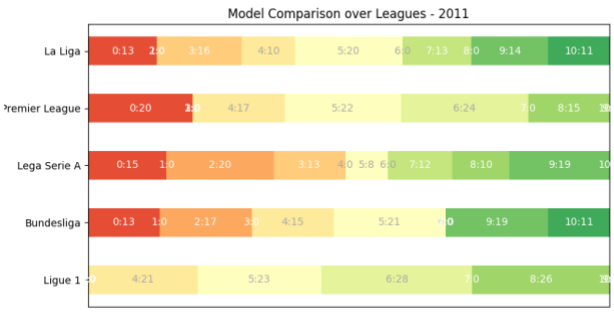
Despite the result is not ideal, I'm going to precede to next experiment. I hope, as experiments going, I will have reasons to explain this.

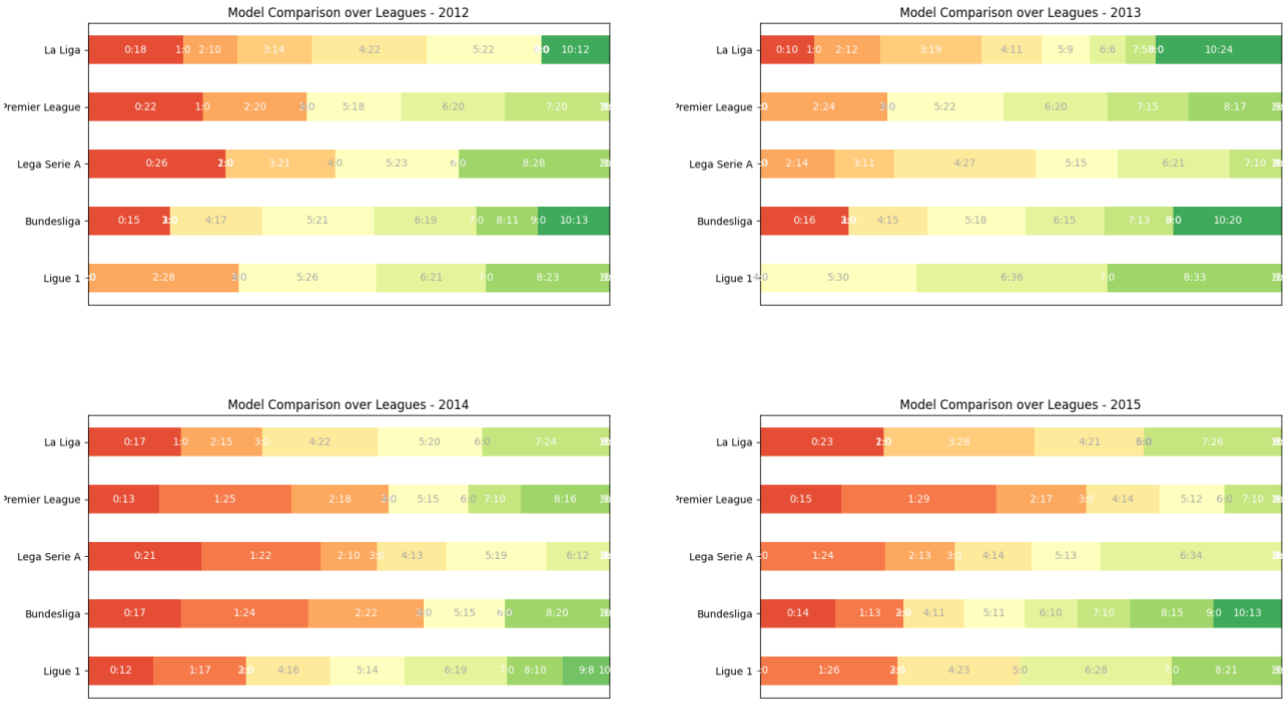
C. Comparison of Different League

As I mentioned before, for major leagues, they all have their unique playing genre (usually it's their national team playing genre, but I don't have data for that). Despite they learn from each other, it's still quite different experiences when watching teams from different leagues.

To evaluate their playing genre, I used the weighted points from section A, which can tell us which feature is more deterministic. I calculated all the points from different league and scale it into percentage. Following are my result:







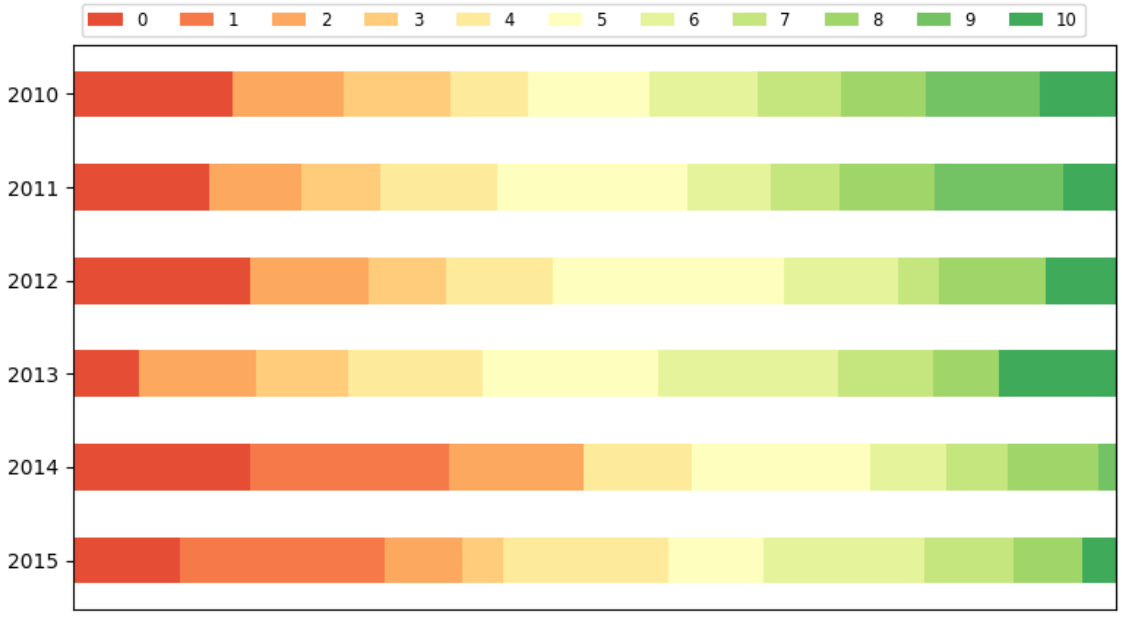
Above are playing genre comparison over five major leagues from 2010 – 2015. For each figure, there are five different leagues. And for each league, I color-coded different features and the text A:B means this is feature #A and it's point take up B percentage. There are some texts on the middle of two color blocks, which means for these features, they are not deterministic at all and they take up 0 percent.

We can clearly see that for different leagues, they have different focus. Take the figure from 2010 for example, we can observe that Premier League focus a lot on #0 where Bundesliga focus a lot on #5. If we take a closer look at what they are, we will know that #0 is speed and #5 is creation crossing. These results make sense because Premier League really focus on speed and body confrontation while Bundesliga focus more on taking attack from two sides and finish with a crossing. I understand for non-soccer fans these may sounds like a make-up, but I have been a soccer fans for over 10 years and these results really is the playing genre for that league in that year.

What really matter here is that despite my prediction accuracy was not so good. The model I have did accurately captured the key features of different leagues. This means that my model actually was right and reasonable.

D. Comparison of Different Year

Similar as section C, I draw out the playing genre of different years based on all five major leagues. Following is my result:

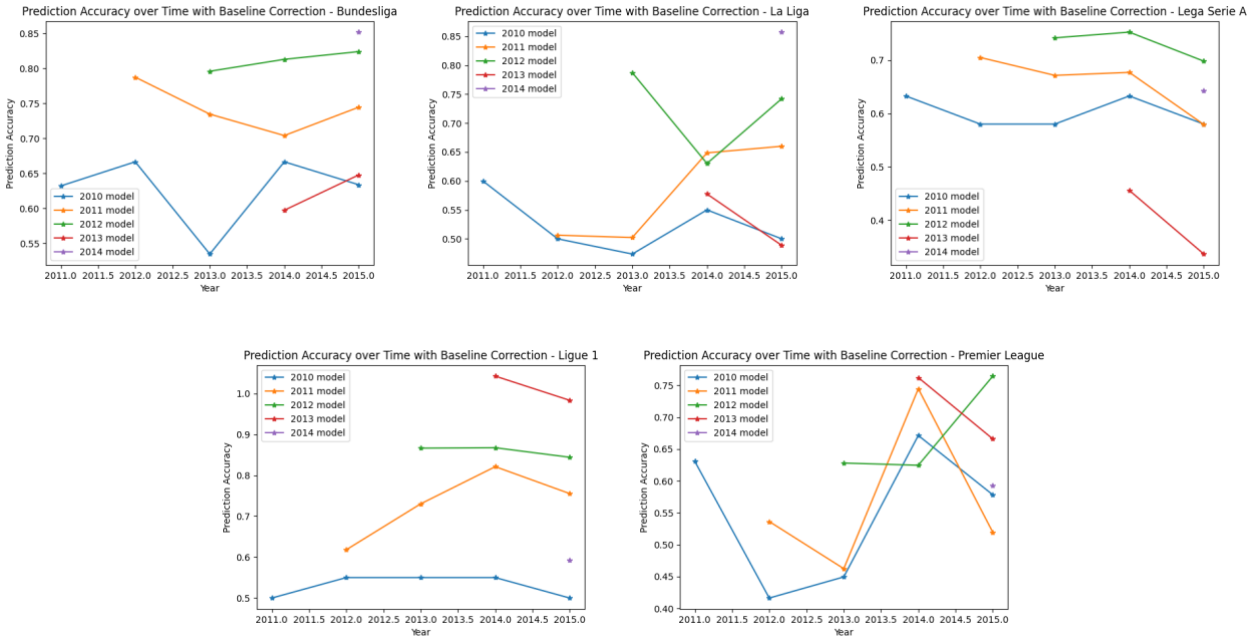


We can see that between 2010 – 2013, clubs in major leagues are focus a lot on #5 - #10, which are related to passing and positioning. This is historical accurate because after FC Barcelona made a huge success in 2009 by controlling the game using passing and positioning, all clubs have been learning from them. However, as FC Barcelona's players got older and older, they were having difficulties to maintain they old way of playing and was defeated by FC Bayern Munich in 2013. FC Bayern Munich focus on speed and dribbling, and we can see that playing genre in 2014 was changed a lot comparing to 2013. In 2014, clubs are focusing on #0 and #1 which are speed and dribbling. These are exactly the way FC Bayern Munich was playing.

My model captured the changing of playing genre over time. And it's historical accurate. These results also provide evidence that my model is correct and meaningful.

E. Further Analysis of Prediction Accuracy

From previous two experiments, I already know that my model can capture the features I want. However, I still cannot explain the low prediction accuracy in section B. Therefore, I'm going to redo that experiment. And for this time, I will focus more on influence of different leagues and year. Following are my results:

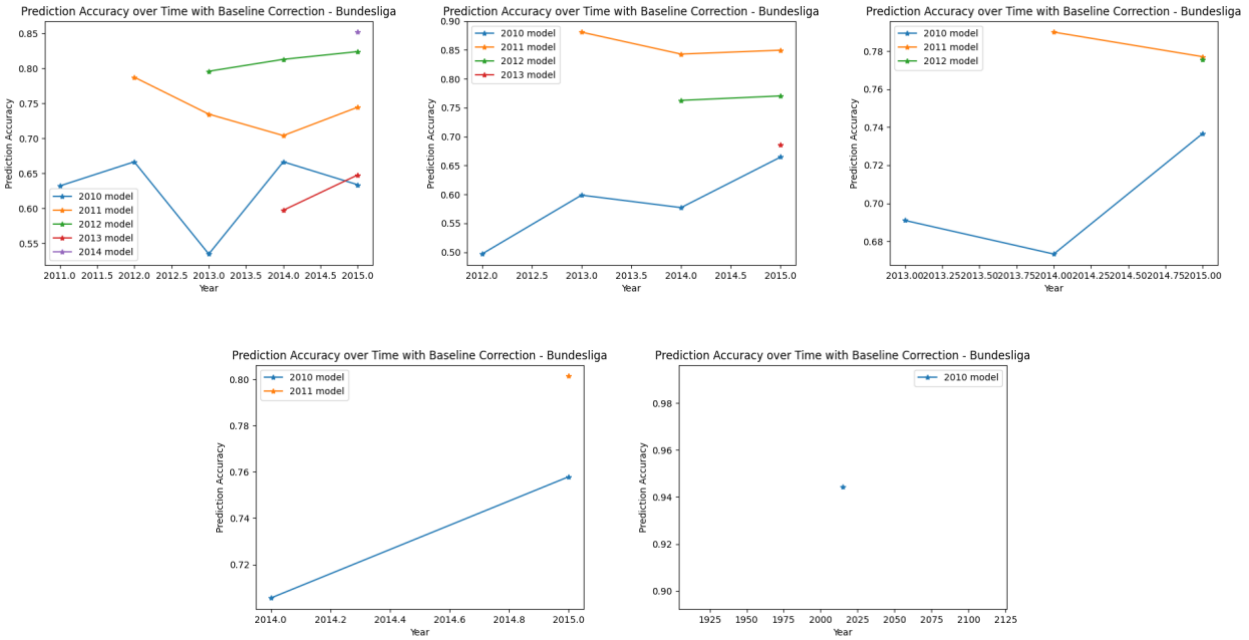


We can see from above figures that there is quite difference of prediction accuracy between different leagues. The La Liga we used in section B is actually the lowest. Comparing to Bundesliga and Ligue 1, the other three leagues are lower. And comparing to others, Premier League is more un-predictable since the figure lines are less smooth. This might indicate why Premier League is the most lucrative league in the world. Because they have the most competitive games and it's really hard to predict the final ranking.

From above, we know that for different leagues, my way of predicting ranking will give different outcome. Particularly, for league with less competition like Bundesliga and Ligue 1, my model will give relatively good results. It makes sense considering my way of prediction is evaluate the playing genre and for very competitive leagues, they are going to change the way they play rapidly to keep their ranking.

What if we train the model not just for 1 year?

I explained above that playing genre is changing over year and to train a model with multiple year will be meaningless. But, can I improve my model by introducing some not very closely related data? Following are my results:



It's quite clear that with more data, I'm having better prediction results. A possible reason for this is that despite the relationship between year is not strong, there are some clubs keep the same way they play and don't change over time. For these clubs, training multiple years means their way of playing and their ranking will be enhanced. After all, for each league, their main playing genre won't change in short time.

6. Conclusion

In this report, I started with a European soccer database and tried to predict the club ranking based on the way they are playing. After that, I defined my problem as a multi-classification problem on discrete data set. My model selection is the decision tree and introduced probability into my model. The overall result is not ideal, but it definitely provide a lot information.

Along my experiments, I analysis results and propose new ideas and experiments. This is actually a very big problem to predict ranking. Because what we are actually predicting is human behavior based on what they previous do as a team. As time goes, team members are changing and their head coach is changing, the financial strength of the club is changing, and the world of soccer is also changing. What I'm doing here is just using a small part of data trying to predict the final result. We know that in the field of machine learning, with mega data, it's basically saying we will have a better result. My model limitation also comes from the data size (and dimensions).

Despite all the problems, my model also predicts and provide a lot of information that is historical accurate. My methods to improve the model was working.

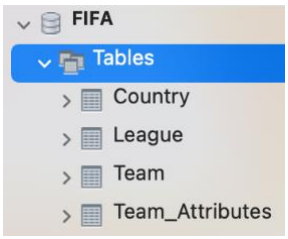
Considering the length of this report, I only include some of my results. And for non-soccer fans, I tried to make this report understandable, but it's not easy or even possible to include soccer strategies in such a short report. The historical information of playing genre I provided in this report was the way I observed, and many observers published over time (genre is more of an opinion instead of concrete math). For this report, we can treat as the ground truth and expected results.

Back to the assignment, I'm asked to come up with a completion agent. For my model, **the prediction I'm doing is the completion**. I have some data which is club playing attributes and I need a completion for their final ranking. When training, if there's one attribute missing, I would just ignore it. When it comes to prediction, if there's data in testing set that haven't showed up in training data, I would predict this data using random pick based on the leaf distributions. My prediction accuracy is the completion benchmark. However, since the problem I'm dealing with have some real-life meaning, therefore I cannot manipulate data any way I want. I need to make sure it doesn't lose the real-life meaning. My model selection is direct and simple. I wanted to focus on the data and analysis instead of showing some fancy models that no one can explain. The result I have, despite not ideal, it's meaningful. And for such a complicated problem, my simple model successfully derived some results, and this prediction/completion is reasonable. In some cases, even accurate.

In conclusion, my model finished the task of prediction/completion and it's getting reasonable results. For some cases, it's showing good results considering how complicated this problem is.

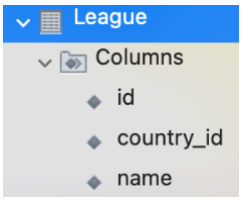
Appendix 1: Database Structure

The database name is FIFA. There are 4 tables in this database. Details are as following:

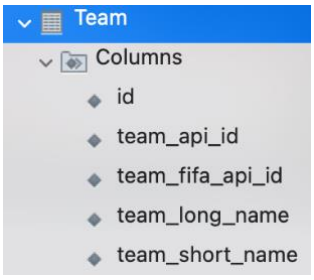


Pic 1. Database Structure

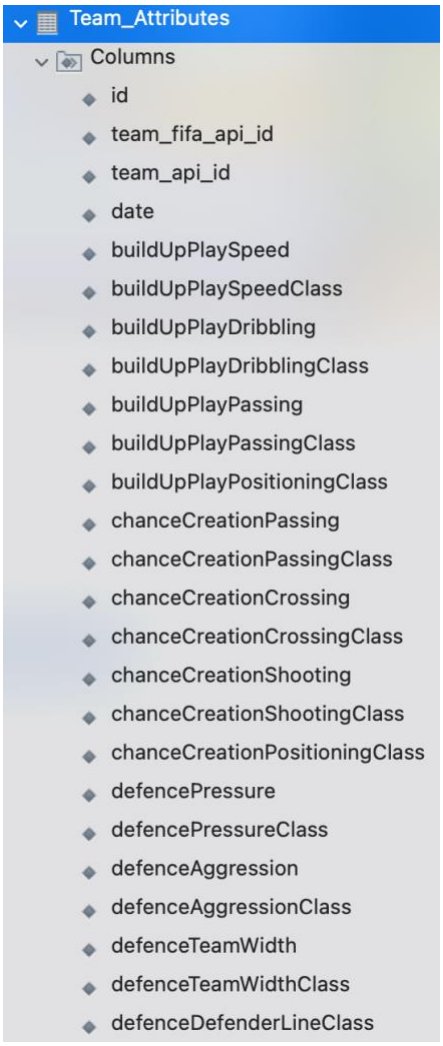
Pic 2. Country Table Structure



Pic 3. League Table Structure



Pic 4. Team Table Structure



Pic 5. Team_Attributes Table Structure

Appendix 2: Source Code (Python 3)

FIFAData.py

```
import mysql.connector


class FIFADB:
    def __init__(self):
        self.cnx = None
        self.attributes = [
            "buildUpPlaySpeedClass",
            "buildUpPlayDribblingClass",
            "buildUpPlayPassingClass",
            "buildUpPlayPositioningClass",
            "chanceCreationPassingClass",
            "chanceCreationCrossingClass",
            "chanceCreationShootingClass",
            "chanceCreationPositioningClass",
            "defencePressureClass",
            "defenceAggressionClass",
            "defenceTeamWidthClass",
            "defenceDefenderLineClass",
        ]
        self.year = {
            2010: "2010-02-22 00:00:00",
            2011: "2011-02-22 00:00:00",
            2012: "2012-02-22 00:00:00",
            2013: "2013-09-20 00:00:00",
            2014: "2014-09-19 00:00:00",
            2015: "2015-09-10 00:00:00",
        }
        self.majorLeagues = [
            "La Liga",
            "Premier League",
```

```
        "Lega Serie A",
        "Bundesliga",
        "Ligue 1",
    ]
    self.Champions = {
        2010: "Inter",
        2011: "FC Barcelona",
        2012: "Chelsea",
        2013: "FC Bayern Munich",
        2014: "Real Madrid CF",
        2015: "FC Barcelona",
    }
    self.LaLiga = {
        2010: [
            "FC Barcelona",
            "Real Madrid CF",
            "Valencia CF",
            "Sevilla FC",
            "RCD Mallorca",
            "Getafe CF",
            "Villarreal CF",
            "Athletic Club de Bilbao",
            "Atlético Madrid",
            "RC Deportivo de La Coruña",
            "RCD Espanyol",
            "CA Osasuna",
            "UD Almería",
            "Real Zaragoza",
            "Real Sporting de Gijón",
            "Racing Santander",
            "Málaga CF",
            "CD Tenerife",
            "Real Valladolid",
            "Xerez Club Deportivo",
        ],
        2011: [
```

```
    "FC Barcelona",
    "Real Madrid CF",
    "Valencia CF",
    "Villarreal CF",
    "Sevilla FC",
    "Athletic Club de Bilbao",
    "Atlético Madrid",
    "RCD Espanyol",
    "CA Osasuna",
    "Real Sporting de Gijón",
    "Málaga CF",
    "Racing Santander",
    "Real Zaragoza",
    "Levante UD",
    "Real Sociedad",
    "Getafe CF",
    "RCD Mallorca",
    "RC Deportivo de La Coruña",
    "Hércules Club de Fútbol",
    "UD Almería",
],
2012: [
    "Real Madrid CF",
    "FC Barcelona",
    "Valencia CF",
    "Málaga CF",
    "Atlético Madrid",
    "Levante UD",
    "CA Osasuna",
    "RCD Mallorca",
    "Sevilla FC",
    "Athletic Club de Bilbao",
    "Getafe CF",
    "Real Sociedad",
    "Real Betis Balompié",
    "RCD Espanyol",
```

```
        "Rayo Vallecano",
        "Real Zaragoza",
        "Granada CF",
        "Villarreal CF",
        "Real Sporting de Gijón",
        "Racing Santander",
    ],
    2013: [
        "FC Barcelona",
        "Real Madrid CF",
        "Atlético Madrid",
        "Real Sociedad",
        "Valencia CF",
        "Málaga CF",
        "Real Betis Balompié",
        "Rayo Vallecano",
        "Sevilla FC",
        "Getafe CF",
        "Levante UD",
        "Athletic Club de Bilbao",
        "RCD Espanyol",
        "Real Valladolid",
        "Granada CF",
        "CA Osasuna",
        "RC Deportivo de La Coruña",
        "RCD Mallorca",
        "Real Zaragoza",
    ],
    2014: [
        "Atlético Madrid",
        "FC Barcelona",
        "Real Madrid CF",
        "Athletic Club de Bilbao",
        "Sevilla FC",
        "Villarreal CF",
        "Real Sociedad",
```

```
    "Valencia CF",
    "RC Celta de Vigo",
    "Levante UD",
    "Málaga CF",
    "Rayo Vallecano",
    "Getafe CF",
    "RCD Espanyol",
    "Granada CF",
    "Elche CF",
    "UD Almería",
    "CA Osasuna",
    "Real Valladolid",
    "Real Betis Balompié",
  ],
  2015: [
    "FC Barcelona",
    "Real Madrid CF",
    "Atlético Madrid",
    "Valencia CF",
    "Sevilla FC",
    "Villarreal CF",
    "Athletic Club de Bilbao",
    "RC Celta de Vigo",
    "Málaga CF",
    "RCD Espanyol",
    "Rayo Vallecano",
    "Real Sociedad",
    "Elche CF",
    "Levante UD",
    "Getafe CF",
    "RC Deportivo de La Coruña",
    "Granada CF",
    "SD Eibar",
    "UD Almería",
    "Córdoba CF",
  ],
```

```
}
self.Premier = {
    2010: [
        "Chelsea",
        "Manchester United",
        "Arsenal",
        "Tottenham Hotspur",
        "Manchester City",
        "Aston Villa",
        "Liverpool",
        "Everton",
        "Birmingham City",
        "Blackburn Rovers",
        "Stoke City",
        "Fulham",
        "Sunderland",
        "Bolton Wanderers",
        "Wolverhampton Wanderers",
        "Wigan Athletic",
        "West Ham United",
        "Burnley",
        "Hull City",
        "Portsmouth",
    ],
    2011: [
        "Manchester United",
        "Chelsea",
        "Manchester City",
        "Arsenal",
        "Tottenham Hotspur",
        "Liverpool",
        "Everton",
        "Fulham",
        "Aston Villa",
        "Sunderland",
        "West Bromwich Albion",
```

```
    "Newcastle United",
    "Stoke City",
    "Bolton Wanderers",
    "Blackburn Rovers",
    "Wigan Athletic",
    "Wolverhampton Wanderers",
    "Birmingham City",
    "Blackpool",
    "West Ham United",
  ],
  2012: [
    "Manchester City",
    "Manchester United",
    "Arsenal",
    "Tottenham Hotspur",
    "Newcastle United",
    "Chelsea",
    "Everton",
    "Liverpool",
    "Fulham",
    "West Bromwich Albion",
    "Swansea City",
    "Norwich City",
    "Sunderland",
    "Stoke City",
    "Wigan Athletic",
    "Aston Villa",
    "Queens Park Rangers",
    "Bolton Wanderers",
    "Blackburn Rovers",
    "Wolverhampton Wanderers",
  ],
  2013: [
    "Manchester United",
    "Manchester City",
    "Chelsea",
```

```
        "Arsenal",
        "Tottenham Hotspur",
        "Everton",
        "Liverpool",
        "West Bromwich Albion",
        "Swansea City",
        "West Ham United",
        "Norwich City",
        "Fulham",
        "Stoke City",
        "Southampton",
        "Aston Villa",
        "Newcastle United",
        "Sunderland",
        "Wigan Athletic",
        "Reading",
        "Queens Park Rangers",
    ],
    2014: [
        "Manchester City",
        "Liverpool",
        "Chelsea",
        "Arsenal",
        "Everton",
        "Tottenham Hotspur",
        "Manchester United",
        "Southampton",
        "Stoke City",
        "Newcastle United",
        "Crystal Palace",
        "Swansea City",
        "West Ham United",
        "Sunderland",
        "Aston Villa",
        "Hull City",
        "West Bromwich Albion",
```

```
            "Norwich City",
            "Fulham",
            "Cardiff City",
        ],
        2015: [
            "Chelsea",
            "Manchester City",
            "Arsenal",
            "Manchester United",
            "Tottenham Hotspur",
            "Liverpool",
            "Southampton",
            "Swansea City",
            "Stoke City",
            "Crystal Palace",
            "Everton",
            "West Ham United",
            "West Bromwich Albion",
            "Leicester City",
            "Newcastle United",
            "Sunderland",
            "Aston Villa",
            "Hull City",
            "Burnley",
            "Queens Park Rangers",
        ],
    }
    self.SerieA = {
        2010: [
            "Inter",
            "Roma",
            "Milan",
            "Sampdoria",
            "Palermo",
            "Napoli",
            "Juventus",
```

```
        "Parma",
        "Genoa",
        "Bari",
        "Fiorentina",
        "Lazio",
        "Catania",
        "Chievo Verona",
        "Udinese",
        "Cagliari",
        "Bologna",
        "Atalanta",
        "Siena",
        "Livorno",
    ],
    2011: [
        "Milan",
        "Inter",
        "Napoli",
        "Udinese",
        "Lazio",
        "Roma",
        "Juventus",
        "Palermo",
        "Fiorentina",
        "Genoa",
        "Chievo Verona",
        "Parma",
        "Catania",
        "Cagliari",
        "Cesena",
        "Bologna",
        "Lecce",
        "Sampdoria",
        "Brescia",
        "Bari",
    ],
```

```
    2012: [
        "Juventus",
        "Milan",
        "Udinese",
        "Lazio",
        "Napoli",
        "Inter",
        "Roma",
        "Parma",
        "Bologna",
        "Chievo Verona",
        "Catania",
        "Atalanta",
        "Fiorentina",
        "Siena",
        "Cagliari",
        "Palermo",
        "Genoa",
        "Lecce",
        "Novara",
        "Cesena",
    ],
    2013: [
        "Juventus",
        "Napoli",
        "Milan",
        "Fiorentina",
        "Udinese",
        "Roma",
        "Lazio",
        "Catania",
        "Inter",
        "Parma",
        "Cagliari",
        "Chievo Verona",
        "Bologna",
```

```
        "Sampdoria",
        "Atalanta",
        "Torino",
        "Genoa",
        "Palermo",
        "Siena",
        "Pescara",
    ],
    2014: [
        "Juventus",
        "Roma",
        "Napoli",
        "Fiorentina",
        "Inter",
        "Parma",
        "Torino",
        "Milan",
        "Lazio",
        "Hellas Verona",
        "Atalanta",
        "Sampdoria",
        "Udinese",
        "Genoa",
        "Cagliari",
        "Chievo Verona",
        "Sassuolo",
        "Catania",
        "Bologna",
        "Livorno",
    ],
    2015: [
        "Juventus",
        "Roma",
        "Lazio",
        "Fiorentina",
        "Napoli",
```

```
            "Genoa",
            "Sampdoria",
            "Inter",
            "Torino",
            "Milan",
            "Palermo",
            "Sassuolo",
            "Hellas Verona",
            "Chievo Verona",
            "Empoli",
            "Udinese",
            "Atalanta",
            "Cagliari",
            "Cesena",
            "Parma",
        ],
    }
    self.Bundesliga = {
        2010: [
            "FC Bayern Munich",
            "FC Schalke 04",
            "SV Werder Bremen",
            "Bayer 04 Leverkusen",
            "Borussia Dortmund",
            "VfB Stuttgart",
            "Hamburger SV",
            "VfL Wolfsburg",
            "1. FSV Mainz 05",
            "Eintracht Frankfurt",
            "TSG 1899 Hoffenheim",
            "Borussia Mönchengladbach",
            "1. FC Köln",
            "SC Freiburg",
            "Hannover 96",
            "1. FC Nürnberg",
            "VfL Bochum",
```

```
    "Hertha BSC Berlin",
],
2011: [
    "Borussia Dortmund",
    "Bayer 04 Leverkusen",
    "FC Bayern Munich",
    "Hannover 96",
    "1. FSV Mainz 05",
    "1. FC Nürnberg",
    "1. FC Kaiserslautern",
    "Hamburger SV",
    "SC Freiburg",
    "1. FC Köln",
    "TSG 1899 Hoffenheim",
    "VfB Stuttgart",
    "SV Werder Bremen",
    "FC Schalke 04",
    "VfL Wolfsburg",
    "Borussia Mönchengladbach",
    "Eintracht Frankfurt",
    "FC St. Pauli",
],
2012: [
    "Borussia Dortmund",
    "FC Bayern Munich",
    "FC Schalke 04",
    "Borussia Mönchengladbach",
    "Bayer 04 Leverkusen",
    "VfB Stuttgart",
    "Hannover 96",
    "VfL Wolfsburg",
    "SV Werder Bremen",
    "1. FC Nürnberg",
    "TSG 1899 Hoffenheim",
    "SC Freiburg",
    "1. FSV Mainz 05",
```

```
    "FC Augsburg",
    "Hamburger SV",
    "Hertha BSC Berlin",
    "1. FC Köln",
    "1. FC Kaiserslautern",
  ],
  2013: [
    "FC Bayern Munich",
    "Borussia Dortmund",
    "Bayer 04 Leverkusen",
    "FC Schalke 04",
    "SC Freiburg",
    "Eintracht Frankfurt",
    "Hamburger SV",
    "Borussia Mönchengladbach",
    "Hannover 96",
    "1. FC Nürnberg",
    "VfL Wolfsburg",
    "VfB Stuttgart",
    "1. FSV Mainz 05",
    "SV Werder Bremen",
    "FC Augsburg",
    "TSG 1899 Hoffenheim",
    "Fortuna Düsseldorf",
    "SpVgg Greuther Fürth",
  ],
  2014: [
    "FC Bayern Munich",
    "Borussia Dortmund",
    "FC Schalke 04",
    "Bayer 04 Leverkusen",
    "VfL Wolfsburg",
    "Borussia Mönchengladbach",
    "1. FSV Mainz 05",
    "FC Augsburg",
    "TSG 1899 Hoffenheim",
```

```
        "Hannover 96",
        "Hertha BSC Berlin",
        "SV Werder Bremen",
        "Eintracht Frankfurt",
        "SC Freiburg",
        "VfB Stuttgart",
        "Hamburger SV",
        "1. FC Nürnberg",
        "Eintracht Braunschweig",
    ],
    2015: [
        "FC Bayern Munich",
        "VfL Wolfsburg",
        "Borussia Mönchengladbach",
        "Bayer 04 Leverkusen",
        "FC Augsburg",
        "FC Schalke 04",
        "Borussia Dortmund",
        "TSG 1899 Hoffenheim",
        "Eintracht Frankfurt",
        "SV Werder Bremen",
        "1. FSV Mainz 05",
        "1. FC Köln",
        "Hannover 96",
        "VfB Stuttgart",
        "Hertha BSC Berlin",
        "Hamburger SV",
        "SC Freiburg",
        "SC Paderborn 07",
    ],
}
self.League1 = {
    2010: [
        "Olympique de Marseille",
        "Olympique Lyonnais",
        "AJ Auxerre",
```

```
    "LOSC Lille",
    "Montpellier Hérault SC",
    "Girondins de Bordeaux",
    "FC Lorient",
    "AS Monaco",
    "Stade Rennais FC",
    "Valenciennes FC",
    "RC Lens",
    "AS Nancy-Lorraine",
    "Paris Saint-Germain",
    "Toulouse FC",
    "OGC Nice",
    "FC Sochaux-Montbéliard",
    "AS Saint-Étienne",
    "Le Mans FC",
    "US Boulogne Cote D'Opale",
    "Grenoble Foot 38",
],
2011: [
    "LOSC Lille",
    "Olympique de Marseille",
    "Olympique Lyonnais",
    "Paris Saint-Germain",
    "FC Sochaux-Montbéliard",
    "Stade Rennais FC",
    "Girondins de Bordeaux",
    "Toulouse FC",
    "AJ Auxerre",
    "AS Saint-Étienne",
    "FC Lorient",
    "Valenciennes FC",
    "AS Nancy-Lorraine",
    "Montpellier Hérault SC",
    "SM Caen",
    "Stade Brestois 29",
    "OGC Nice",
```

```
    "AS Monaco",
    "RC Lens",
    "AC Arles-Avignon",
  ],
  2012: [
    "Montpellier Hérault SC",
    "Paris Saint-Germain",
    "LOSC Lille",
    "Olympique Lyonnais",
    "Girondins de Bordeaux",
    "Stade Rennais FC",
    "AS Saint-Étienne",
    "Toulouse FC",
    "Évian Thonon Gaillard FC",
    "Olympique de Marseille",
    "AS Nancy-Lorraine",
    "Valenciennes FC",
    "OGC Nice",
    "FC Sochaux-Montbéliard",
    "Stade Brestois 29",
    "AC Ajaccio",
    "FC Lorient",
    "SM Caen",
    "Dijon FCO",
    "AJ Auxerre",
  ],
  2013: [
    "Paris Saint-Germain",
    "Olympique de Marseille",
    "Olympique Lyonnais",
    "OGC Nice",
    "AS Saint-Étienne",
    "LOSC Lille",
    "Girondins de Bordeaux",
    "FC Lorient",
    "Montpellier Hérault SC",
```

```
    "Toulouse FC",
    "Valenciennes FC",
    "SC Bastia",
    "Stade Rennais FC",
    "Stade de Reims",
    "FC Sochaux-Montbéliard",
    "Évian Thonon Gaillard FC",
    "AC Ajaccio",
    "AS Nancy-Lorraine",
    "ES Troyes AC",
    "Stade Brestois 29",
  ],
  2014: [
    "Paris Saint-Germain",
    "AS Monaco",
    "LOSC Lille",
    "AS Saint-Étienne",
    "Olympique Lyonnais",
    "Olympique de Marseille",
    "Girondins de Bordeaux",
    "FC Lorient",
    "Toulouse FC",
    "SC Bastia",
    "Stade de Reims",
    "Stade Rennais FC",
    "FC Nantes",
    "Évian Thonon Gaillard FC",
    "Montpellier Hérault SC",
    "En Avant de Guingamp",
    "OGC Nice",
    "FC Sochaux-Montbéliard",
    "Valenciennes FC",
    "AC Ajaccio",
  ],
  2015: [
    "Paris Saint-Germain",
```

```
            "Olympique Lyonnais",
            "AS Monaco",
            "Olympique de Marseille",
            "AS Saint-Étienne",
            "Girondins de Bordeaux",
            "Montpellier Hérault SC",
            "LOSC Lille",
            "Stade Rennais FC",
            "En Avant de Guingamp",
            "OGC Nice",
            "SC Bastia",
            "SM Caen",
            "FC Nantes",
            "Stade de Reims",
            "FC Lorient",
            "Toulouse FC",
            "Évian Thonon Gaillard FC",
            "FC Metz",
            "RC Lens",
        ],
    }
    self.leagueTrans = {
        "La Liga": self.LaLiga,
        "Premier League": self.Premier,
        "Lega Serie A": self.SerieA,
        "Bundesliga": self.Bundesliga,
        "Ligue 1": self.League1,
    }

def startConnection(self):
    self.cnx = mysql.connector.connect(
        user="root", password="password4FIFA", host="35.194.63.144", database="FIFA"
    )

def closeConnection(self):
    self.cnx.close()
```

```
def getTeamIDbyName(self, teamName):
    if not self.cnx:
        return None
    cursor = self.cnx.cursor()
    query = "SELECT team_fifa_api_id FROM Team WHERE team_long_name = %s"
    cursor.execute(query, (teamName,))
    result = cursor.fetchall()
    return result[0][0]

def getTeamAttributebyIDandYear(self, id, year):
    if not self.cnx:
        return None
    cursor = self.cnx.cursor()
    attrs = ""
    for attr in self.attributes:
        attrs += attr + ", "
    attrs = attrs[:-2]
    query = (
        "SELECT "
        + attrs
        + " FROM Team_Attributes WHERE team_fifa_api_id = %s AND date = %s"
    )
    cursor.execute(
        query,
        (
            id,
            self.year[year],
        ),
    )
    result = cursor.fetchall()
    return list(result[0][:-1])

def getLeagueDatabyYear(self, year, league):
    X = []
    Y = []
```

```
        rank = 1
        for teamName in self.leagueTrans[league][year]:
            teamID = self.getTeamIDbyName(teamName)
            data = self.getTeamAttributebyIDandYear(teamID, year)
            X.append(data)
            if rank <= 6:
                Y.append("TOP")
            elif rank <= 17:
                Y.append("MIDDLE")
            else:
                Y.append("TAIL")
            rank += 1
        return X, Y
```

DecisionTreeForFIFA.py

```
from math import log2
from collections import Counter, defaultdict
from random import random


class TreeNode:
    def __init__(self, measurement=None):
        self.measurement = measurement
        self.result = {}
        self.branch = {}
        # self.levelPrediction = [None, None]
        self.isEnd = False
        self.prob = []
        self.ctrRemain = None


class DecisionTree:
    def __init__(self, data, Y):
        self.data = data
        self.Y = Y
```

```
    self.root = self.ID3(TreeNode(), self.data, self.Y, list(range(len(data[0]))))
    self.level = self.getMeasurementbyLevel()


def calcIG(self, data, Y, features):
    ig = []
    ctrY = Counter(Y)
    # Try to avoid 'math domain error'
    # yEntropy = ((yOnes/len(Y))*log2(yOnes/len(Y)) + (yZeros/len(Y))*log2(yZeros/len(Y))) * -1
    yEntropy = 0
    for y in ctrY.values():
        if y > 0:
            yEntropy += (y / len(Y)) * log2(y / len(Y))
    yEntropy *= -1
    for j in range(len(data[0])):
        if j not in features:
            ig.append(0)
            continue
        featureVal = defaultdict(list)
        entropy = []
        featureLen = []
        for i in range(len(data)):
            # print('i:', i, ', j:', j)
            featureVal[data[i][j]].append(i)
        for vals in featureVal:
            valY = defaultdict(int)
            for v in featureVal[vals]:
                valY[Y[v]] += 1
            # Try to avoid 'math domain error'
            curEntropy = 0
            for valYElem in valY.keys():
                if valY[valYElem] > 0:
                    curEntropy += (valY[valYElem] / len(featureVal[vals])) * log2(
                        valY[valYElem] / len(featureVal[vals])
                    )
            curEntropy *= -1
            # curEntropy = ((valY[0]/len(vals))*log2(valY[0]/len(vals)) + (valY[1]/len(vals))*log2(valY[1]/len(vals))) * -1
```

```
            entropy.append(curEntropy)
            featureLen.append(len(featureVal[vals]))
        # curIG = yEntropy - ((len(featureVal[0])/len(data))*entropy[0] + (len(featureVal[1])/len(data))*entropy[1])
        curIG = yEntropy - sum(
            [featureLen[i] / len(data) * entropy[i] for i in range(len(featureVal))]
        )
        ig.append(curIG)
    return ig

def separateData(self, data, Y, pivot):
    pivotDict = defaultdict(list)
    for i in range(len(data)):
        pivotDict[data[i][pivot]].append(i)
    datas = []
    ys = []
    for measurement in pivotDict.keys():
        d = []
        y = []
        for index in pivotDict[measurement]:
            tmp = []
            for j in range(len(data[0])):
                tmp.append(data[index][j])
            d.append(tmp)
            y.append(Y[index])
        datas.append(d)
        ys.append(y)
    return datas, ys

def allSame(self, Y):
    if not Y:
        return True
    for i in range(1, len(Y)):
        if Y[0] != Y[i]:
            return False
    return True
```

```
def ID3(self, node, data, Y, features):
    node.ctrRemain = Counter(Y)
    ig = self.calcIG(data, Y, features)
    pivot = ig.index(max(ig))
    if pivot == 0 and max(ig) == 0:
        node.measurement = "Prob"
        ctrY = Counter(Y)
        tmp = 0
        for keyY in ctrY:
            node.prob.append([ctrY[keyY] / len(Y) + tmp, keyY])
            tmp = node.prob[-1][0]
        # print("training prob:", node.prob)
        return node
    node.measurement = pivot
    Xs, Ys = self.separateData(data, Y, pivot)
    nxtFeatures = features[:]
    nxtFeatures.remove(pivot)
    # node.levelPrediction[0] = len(Y) - sum(Y)
    # node.levelPrediction[1] = sum(Y)
    # if len(data) <= self.sizeThreshold:
    #     node.isEnd = True
    # print('node.measurement:', node.measurement, 'Y:', Y)
    for i in range(len(Xs)):
        if self.allSame(Ys[i]):
            node.result[Xs[i][0][pivot]] = Ys[i][0]
        else:
            node.branch[Xs[i][0][pivot]] = self.ID3(
                TreeNode(), Xs[i], Ys[i], nxtFeatures
            )
    return node


def printTree(self):
    queue = [[self.root.measurement, self.root]]
    while queue:
        tmp = []
        print("--- New Level ---")
```

```
        for name, node in queue:
            print("feature #:", name)
            if node.result:
                print("result:", node.result)
            if node.prob:
                print("prob:", node.prob)
            if node.branch:
                for i in node.branch.keys():
                    tmp.append([node.branch[i].measurement, node.branch[i]])
        queue = tmp

def getMeasurementbyLevel(self):
    queue = [[self.root.measurement, self.root]]
    ans = []
    while queue:
        tmp = []
        curLevel = []
        for name, node in queue:
            curLevel.append(name)
            if node.branch:
                for i in node.branch.keys():
                    tmp.append([node.branch[i].measurement, node.branch[i]])
        ans.append(curLevel)
        queue = tmp
    return ans

def predict(self, testData):
    if not self.root:
        print("ERROR: Train the model first!")
        return
    node = self.root
    while node:
        if node.measurement == "Prob":
            cur = "Prob"
        else:
            cur = testData[node.measurement]
```

```
        if cur in node.result:
            return node.result[cur]
        elif cur in node.branch:
            node = node.branch[cur]
        elif cur == "Prob":
            seed = random()
            i = 0
            while seed >= node.prob[i][0]:
                i += 1
            return node.prob[i][1]
        else:
            seed = random()
            tmp = 0
            total = sum(list(node.ctrRemain.values()))
            for k in node.ctrRemain:
                tmp += node.ctrRemain[k] / total
                if seed <= tmp:
                    return k

def calcError(self, data, Y):
    errorCtr = 0
    for i in range(len(data)):
        if self.predict(data[i]) != Y[i]:
            errorCtr += 1
    return errorCtr / len(data)

def calcMatchRate(self, data, Y):
    if not self.root:
        print("ERROR: Train the model first!")
        return
    L = len(Y)
    errorNum = 0
    for i in range(L):
        errorNum += 1 if Y[i] != self.predict(data[i]) else 0
    return (L - errorNum) / L
```

Final.py

```
from FIFAData import FIFADB
from DecisionTreeForFIFA import DecisionTree
import matplotlib.pyplot as plt
import numpy as np


class Visualization:
    def plotHistogram(self, X, xLable, yLable, league, year):
        plt.clf()
        title = "Histogram of " + league + " in " + str(year)
        plt.hist(X, range=(0, 1))
        plt.xlabel(xLable)
        plt.ylabel(yLable)
        plt.title(title)
        fileName = "Hist-" + league + "-" + str(year) + ".png"
        plt.savefig(fileName)

    def plotDots(self, X, Y, xLable, yLable, title):
        plt.clf()
        plt.scatter(X, Y)
        plt.title(title)
        plt.xlabel(xLable)
        plt.ylabel(yLable)
        fileName = "Dots-" + title + ".png"
        plt.savefig(fileName)


if __name__ == "__main__":
    # Initiate Visualization Tools
    v = Visualization()

    # # Usage Demo
    # # Initiate Database
    # db = FIFADB()
```

```
# # Define the year, year range: 2010 - 2015
# year = 2012
# # Define the league, league names are: ["La Liga", "Premier League", "Lega Serie A", "Bundesliga", "Ligue 1"]
# league = 'La Liga'
# # Start Database Connection
# db.startConnection()
# # Take data as the way we need
# X, Y = db.getLeagueDatabyYear(year, league)
# # Close Database Connection
# db.closeConnection()
# print("X:")
# for xx in X:
#     print(xx)
# print("Y:")
# for yy in Y:
#     print(yy)
# # Train the model
# model = DecisionTree(X, Y)
# # Print out the model structure
# model.printTree()
# # Calculate and print accuracy of model
# print("Accuracy of Model:", model.calcMatchRate(X, Y))

# # A. Evaluation of Model
# league = "La Liga"
# db = FIFADB()
# db.startConnection()
# # X, Y = db.getLeagueDatabyYear(2012, league)
# # print("X:")
# # for xx in X:
# #     print(xx)
# # print("Y:")
# # for yy in Y:
# #     print(yy)
# # model = DecisionTree(X, Y)
# # model.printTree()
```

```
# # for year in range(2010, 2016):
# #     X, Y = db.getLeagueDatabyYear(year, league)
# #     model = DecisionTree(X, Y)
# #     matchRateList = []
# #     for _ in range(50):
# #         matchRateList.append(model.calcMatchRate(X, Y))
# #     v.plotHistogram(
# #         matchRateList, "Accuracy of Model", "Number of Instance", league, year
# #     )
# plt.clf()
# points = [0 for _ in range(11)]
# for year in range(2010, 2016):
#     curPoints = [0 for _ in range(11)]
#     X, Y = db.getLeagueDatabyYear(year, league)
#     model = DecisionTree(X, Y)
#     print("Measurement by Level for " + str(year) + ":", model.level)
#     for i in range(len(model.level)):
#         for measurement in model.level[i]:
#             if measurement == "Prob":
#                 continue
#             points[measurement] += 11 - i
#             curPoints[measurement] += 11 - i
#     plt.plot(list(range(11)), curPoints, "-*", label=str(year))
#     print("curPoints:", curPoints)
# plt.scatter(list(range(11)), points, marker=(5, 1), label="Total")
# plt.legend()
# plt.title("Points vs Attributes")
# plt.xlabel("Attributes")
# plt.ylabel("Points")
# fileName = "Points-Attributes.png"
# plt.savefig(fileName)
# db.closeConnection()

# B. Ranking Prediction
# plt.clf()
# league = "La Liga"
```

```
# numRound = 100
# db = FIFADB()
# db.startConnection()
# for year in range(2010, 2015):
#     trainX, trainY = db.getLeagueDatabyYear(year, league)
#     model = DecisionTree(trainX, trainY)
#     correctRate = []
#     for testYear in range(year+1, 2016):
#         testX, testY = db.getLeagueDatabyYear(testYear, league)
#         totalRate = 0
#         for _ in range(numRound):
#             numCorrect = 0
#             for j in range(len(testY)):
#                 numCorrect += 1 if testY[j] == model.predict(testX[j]) else 0
#             totalRate += numCorrect / len(testY)
#         correctRate.append(totalRate / numRound)
#     plt.plot(list(range(year+1, 2016, 1)), correctRate, "-*", label=str(year)+" model")
# plt.legend()
# plt.title("Prediction Accuracy over Time")
# plt.xlabel("Year")
# plt.ylabel("Prediction Accuracy")
# fileName = "PredictionAccuracyOverTime-" + league + ".png"
# plt.savefig(fileName)
# db.closeConnection()

# plt.clf()
# league = "La Liga"
# numRound = 100
# db = FIFADB()
# db.startConnection()
# for year in range(2010, 2015):
#     trainX, trainY = db.getLeagueDatabyYear(year, league)
#     model = DecisionTree(trainX, trainY)
#     baseLineRate = 0
#     for _ in range(numRound):
#         numCorrectBase = 0
```

```
#         for j in range(len(trainY)):
#             numCorrectBase += 1 if trainY[j] == model.predict(trainX[j]) else 0
#         baseLineRate += numCorrectBase / len(trainY)
#     baseLine = baseLineRate / numRound
#     correctRate = []
#     for testYear in range(year+1, 2016):
#         testX, testY = db.getLeagueDatabyYear(testYear, league)
#         totalRate = 0
#         for _ in range(numRound):
#             numCorrect = 0
#             for j in range(len(testY)):
#                 numCorrect += 1 if testY[j] == model.predict(testX[j]) else 0
#             totalRate += numCorrect / len(testY)
#         correctRate.append(totalRate / numRound / baseLine)
#     plt.plot(list(range(year+1, 2016, 1)), correctRate, "-*", label=str(year)+" model")
# plt.legend()
# plt.title("Prediction Accuracy over Time with Baseline Correction")
# plt.xlabel("Year")
# plt.ylabel("Prediction Accuracy")
# fileName = "PredictionAccuracyOverTimeWithBaselineCorrection-" + league + ".png"
# plt.savefig(fileName)
# db.closeConnection()

# # C. Comparison of Different League
# plt.clf()
# year = 2015
# numRound = 100
# db = FIFADB()
# db.startConnection()
# scaledPoints = []
# for league in db.majorLeagues:
#     trainX, trainY = db.getLeagueDatabyYear(year, league)
#     model = DecisionTree(trainX, trainY)
#     points = [0 for _ in range(11)]
#     for i in range(len(model.level)):
#         for measurement in model.level[i]:
```

```
#             if measurement == "Prob":
#                 continue
#             points[measurement] += 11 - i
#     scaledPoints.append([p/sum(points)*100 for p in points])

# print("scalePoints:", scaledPoints)
# category_names = list(range(11))
# labels = db.majorLeagues
# data = np.array(scaledPoints)
# data_cum = data.cumsum(axis=1)
# category_colors = plt.get_cmap('RdYlGn')(
#     np.linspace(0.15, 0.85, data.shape[1]))

# fig, ax = plt.subplots(figsize=(9.2, 5))
# ax.invert_yaxis()
# ax.xaxis.set_visible(False)
# ax.set_xlim(0, np.sum(data, axis=1).max())

# for i, (colname, color) in enumerate(zip(category_names, category_colors)):
#     widths = data[:, i]
#     starts = data_cum[:, i] - widths
#     ax.barh(labels, widths, left=starts, height=0.5,
#             label=colname, color=color)
#     xcenters = starts + widths / 2

#     r, g, b, _ = color
#     text_color = 'white' if r * g * b < 0.5 else 'darkgrey'
#     for y, (x, c) in enumerate(zip(xcenters, widths)):
#         ax.text(x, y, str(i) + ":" + str(int(c)), ha='center', va='center',
#                 color=text_color)
# # ax.legend(ncol=len(category_names), bbox_to_anchor=(0, 1),
# #           loc='lower left', fontsize='small')

# title = "Model Comparison over Leagues - " + str(year)
# plt.title(title)
# fileName = "ModelComparisonOverLeagues-" + str(year) + ".png"
```

```
# plt.savefig(fileName)
# db.closeConnection()

# # D. Comparison of Different Year
# plt.clf()
# numRound = 100
# db = FIFADB()
# db.startConnection()
# scaledPoints = []

# for year in range(2010, 2016):
#     points = [0 for _ in range(11)]
#     for league in db.majorLeagues:
#         trainX, trainY = db.getLeagueDatabyYear(year, league)
#         model = DecisionTree(trainX, trainY)
#         for i in range(len(model.level)):
#             for measurement in model.level[i]:
#                 if measurement == "Prob":
#                     continue
#                 points[measurement] += 11 - i
#     scaledPoints.append([p/sum(points)*100 for p in points])

# print("scalePoints:", scaledPoints)
# category_names = list(range(11))
# labels = list(range(2010, 2016))
# data = np.array(scaledPoints)
# data_cum = data.cumsum(axis=1)
# category_colors = plt.get_cmap('RdYlGn')(
#     np.linspace(0.15, 0.85, data.shape[1]))

# fig, ax = plt.subplots(figsize=(9.2, 5))
# ax.invert_yaxis()
# ax.xaxis.set_visible(False)
# ax.set_xlim(0, np.sum(data, axis=1).max())

# for i, (colname, color) in enumerate(zip(category_names, category_colors)):
```

```
#     widths = data[:, i]
#     starts = data_cum[:, i] - widths
#     ax.barh(labels, widths, left=starts, height=0.5,
#             label=colname, color=color)
#     xcenters = starts + widths / 2

#     r, g, b, _ = color
#     text_color = 'white' if r * g * b < 0.5 else 'darkgrey'
#     for y, (x, c) in enumerate(zip(xcenters, widths)):
#         ax.text(x, y, str(i) + ":" + str(int(c)), ha='center', va='center',
#                 color=text_color)
# ax.legend(ncol=len(category_names), bbox_to_anchor=(0, 1),
#           loc='lower left', fontsize='small')

# # title = "Model Comparison over Year"
# # plt.title(title)
# fileName = "ModelComparisonOverYear.png"
# plt.savefig(fileName)
# db.closeConnection()

# # E. Genre of the Year
# plt.clf()
# db = FIFADB()
# db.startConnection()
# modelsByYear = []

# for year in range(2010, 2016):
#     trainX = []
#     trainY = []
#     for league in db.majorLeagues:
#         trainLeagueX, trainLeagueY = db.getLeagueDatabyYear(year, league)
#         trainX.extend(trainLeagueX)
#         trainY.extend(trainLeagueY)
#     modelsByYear.append(DecisionTree(trainX, trainY))
#     print(str(year) + " : ", modelsByYear[-1].level)
```

```
# db.closeConnection()


# # Further. B
# plt.clf()
# numRound = 100
# db = FIFADB()
# db.startConnection()
# for steps in range(1, 6):
#     for league in db.majorLeagues:
#         plt.clf()
#         for year in range(2010, 2016-steps):
#             trainX = []
#             trainY = []
#             for ss in range(steps):
#                 tx, ty = db.getLeagueDatabyYear(year+ss, league)
#                 trainX.extend(tx)
#                 trainY.extend(ty)
#             model = DecisionTree(trainX, trainY)
#             baseLineRate = 0
#             for _ in range(numRound):
#                 numCorrectBase = 0
#                 for j in range(len(trainY)):
#                     numCorrectBase += 1 if trainY[j] == model.predict(trainX[j]) else 0
#                 baseLineRate += numCorrectBase / len(trainY)
#             baseLine = baseLineRate / numRound
#             correctRate = []
#             for testYear in range(year+steps, 2016):
#                 testX, testY = db.getLeagueDatabyYear(testYear, league)
#                 totalRate = 0
#                 for _ in range(numRound):
#                     numCorrect = 0
#                     for j in range(len(testY)):
#                         numCorrect += 1 if testY[j] == model.predict(testX[j]) else 0
#                     totalRate += numCorrect / len(testY)
#                 correctRate.append(totalRate / numRound / baseLine)
#             plt.plot(list(range(year+steps, 2016, 1)), correctRate, "-*", label=str(year)+" model")
```

```
#         plt.legend()
#         title = "Prediction Accuracy over Time with Baseline Correction - " + league
#         plt.title(title)
#         plt.xlabel("Year")
#         plt.ylabel("Prediction Accuracy")
#         fileName = "PredictionAccuracyOverTimeWithBaselineCorrection-" + str(steps) + "-" + league + ".png"
#         plt.savefig(fileName)
# db.closeConnection()

# Further. B Reversed
plt.clf()
numRound = 100
db = FIFADB()
db.startConnection()
for steps in range(1, 6):
    for league in db.majorLeagues:
        plt.clf()
        for year in range(2015 - steps, 2009, -1):
            trainX = []
            trainY = []
            for ss in range(steps):
                tx, ty = db.getLeagueDatabyYear(year - ss, league)
                trainX.extend(tx)
                trainY.extend(ty)
            model = DecisionTree(trainX, trainY)
            baseLineRate = 0
            for _ in range(numRound):
                numCorrectBase = 0
                for j in range(len(trainY)):
                    numCorrectBase += (
                        1 if trainY[j] == model.predict(trainX[j]) else 0
                    )
                baseLineRate += numCorrectBase / len(trainY)
            baseLine = baseLineRate / numRound
            correctRate = []
            for testYear in range(2015, year - steps, -1):
```

```
            testX, testY = db.getLeagueDatabyYear(testYear, league)
            totalRate = 0
            for _ in range(numRound):
                numCorrect = 0
                for j in range(len(testY)):
                    numCorrect += (
                        1 if testY[j] == model.predict(testX[j]) else 0
                    )
                totalRate += numCorrect / len(testY)
            correctRate.append(totalRate / numRound / baseLine)
        print(list(range(2015, year - steps, -1)))
        print(correctRate)
        plt.plot(
            list(range(2015, year - steps, -1)),
            correctRate,
            "-*",
            label=str(year) + " model",
        )
    plt.legend()
    title = (
        "Prediction Accuracy over Time with Baseline Correction - "
        + league
        + " - Reversed"
    )
    plt.title(title)
    plt.xlabel("Year")
    plt.ylabel("Prediction Accuracy")
    fileName = (
        "PredictionAccuracyOverTimeWithBaselineCorrection-"
        + str(steps)
        + "-"
        + league
        + "-Reversed.png"
    )
    plt.savefig(fileName)
db.closeConnection()
```