Predicting the Outcome of Soccer Matches

Ayaan Kazerouni and Harsh Patel

1 Problem Statement

Soccer is a widely played sport, and is arguably the only globally played sport. We thought it would be interesting to develop a model that could predict the winner of a soccer game, a task called the 'holy grail' by the provider of the dataset. Our problem statement is formally defined as follows:

We have access to a dataset that has featrures from matches already played, ranging from the number of goals scored by home team and away team to different attributes of the players of individual teams. Given this dataset, our goal is to build a model that predicts the winner of the match fairly accurately. As with a lot of parameters in data analytics and machine learning, 'fairly accurately' is a vague term, but we define it as anything better than 53%, which is the accuracy achieved by the dataset provider (and the target accuracy we mentioned in our proposal). We are unaware of the features used in their model or the nature of their model. We have also not looked at what others have done with this publicly available dataset. The work presented in this report is solely our own.

The following sections present a statistical description of the data and its magnitude, interesting insights from exploration of the data, and results from various data mining algorithms used. The document represents the iterative process we used in developing the best possible classifier for predicting the outcome of a soccer match. We implemented decision trees, logistic regressions, a multinomial regression, and made an attempt at a meta-classifier that combined two weak classifiers.

2 Data Description

We have chosen a European soccer dataset [2], the nature of which as described by the provider is:

- +25,000 matches
- +10,000 playres
- 11 European countries with their lead championship
- Seasons 2008 to 2016
- Players and Teams' attributes sourced from EA Sports' FIFA video game series, including the weekly updates

- Team line up with squad formation (X, Y coordinates)
- Betting odds from up to 10 providers
- Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches

We are thankful to the data provider for consolidating and organizing the data from various online sources. This has reduced work on finding relevant data on our part considerably.

The data is provided in a relational database (SQLite3) of *Matches*, *Players*, *Player-Attributes*, *Teams*, *Team-Attributes*, *Leagues*, and *Countries*. A description of the magnitude of the data is provided in Table 1.

	Count	**Dimensions**
Match	25979	115
Teams	299	5
Team_Attributes	1458	25
Player	11060	7
Player_Attributes	183978	42
Country	11	2
League	11	3

Table 1: Magnitude of the available data

Here, the active reader would have noticed that there are only 299 teams but 1458 Team_Attribute rows. This is because the data provider has provided the attributes for some teams over a span of several years and intuitively enough, the attributes keep changing over time. The same is true for the player attributes.

Some descriptive statistics for team attributes can be seen in table 2.

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
buildUpPlaySpeed	20.00	45.00	52.00	52.46	62.00	80.00
buildUpPlayPassing	20.00	40.00	50.00	48.49	55.00	80.00
chanceCreationPassing	21.00	46.00	52.00	52.17	59.00	80.00
chanceCreationCrossing	20.00	47.00	53.00	53.73	62.00	80.00
chanceCreationShooting	22.00	48.00	53.00	53.97	61.00	80.00
defencePressure	23.00	39.00	45.00	46.02	51.00	72.00
defenceAggression	24.00	44.00	48.00	49.25	55.00	72.00
defenceTeamWidth	29.00	47.00	52.00	52.19	58.00	73.00

Table 2: 5-number summaries for team attributes

2.1 Data Exploration

With our objective in mind, the following two tables were of interest to us:

- Match
- Team_Attributes

Although Player_Attributes might give useful information about the strengths of the team, we haven't delved much into it as there are numerous attributes of the players and for each match, we would have to come up with a normalized attribute for the whole team. This would have complicated our model and we aimed at keeping the model as simple as possible to avoid overfitting.

From the structure of the data, it is apparent that the rest of the tables, except player attributes, are used for indexing purposes (eg. foreign keys from leagues to countries). These data entries, being nominal attributes, are just used to differentiate between two different entities and hence do not contribute towards finding out interesting patterns in the data.

2.2 K-means Clustering

K-Means clustering on team attributes provided some interesting insights. First, we needed to pick a value for k. To do this, we followed [3]. We performed k-means clustering for different values of k from 2 to 12, and plotted the SSE for each k. Our dataset had $n =$ 25000, so this process was relatively inexpensive in R. The decrease in SSE from 2 to 3 was large but didn't change as much past that. This told us that $k = 3$ was an optimum number of clusters. The plot is in figure 1.

Once we had selected k, we performed k-means clustering on the team attributes for each year in 2010 to 2015. To visualize the clustered team attributes, we performed Principal Component Analysis (PCA) to reduce the dimensionality from 8 to 2. The PCA-reduced clusters are in figure 2. We can see more variability between teams in 2010 than in subsequent years. In other words, given two teams, it would be easier to categorize them in 2010 – which presumably means that it would be easier to determine the outcome between two teams in different clusters. For years 2011, the clusters are denser and closer together for the most part, indicating that teams were more closely matched up in 2011. The remaining years see the teams moving further apart again, but not as much as 2010.

Please note that the clustering took place on the original data. PCA was used for visualization purposes only. We also performed clustering on all team attributes across the six years. The centroids gleaned are in table 3.

	C1	C2	C3
buildUpPlaySpeed	57.55	56.53	41.69
buildUpPlayPassing	48.81	53.75	39.61
chanceCreationPassing	60.03	52.65	45.01
chanceCreationCrossing	56.65	55.97	47.71
chanceCreationShooting	60.99	50.98	53.20
defencePressure	56.63	41.66	44.57
defenceAggression	56.11	47.43	46.69
defenceTeamWidth	59.92	49.49	50.34

Table 3: Cluster centers for team attributes from 2010 to 2015.

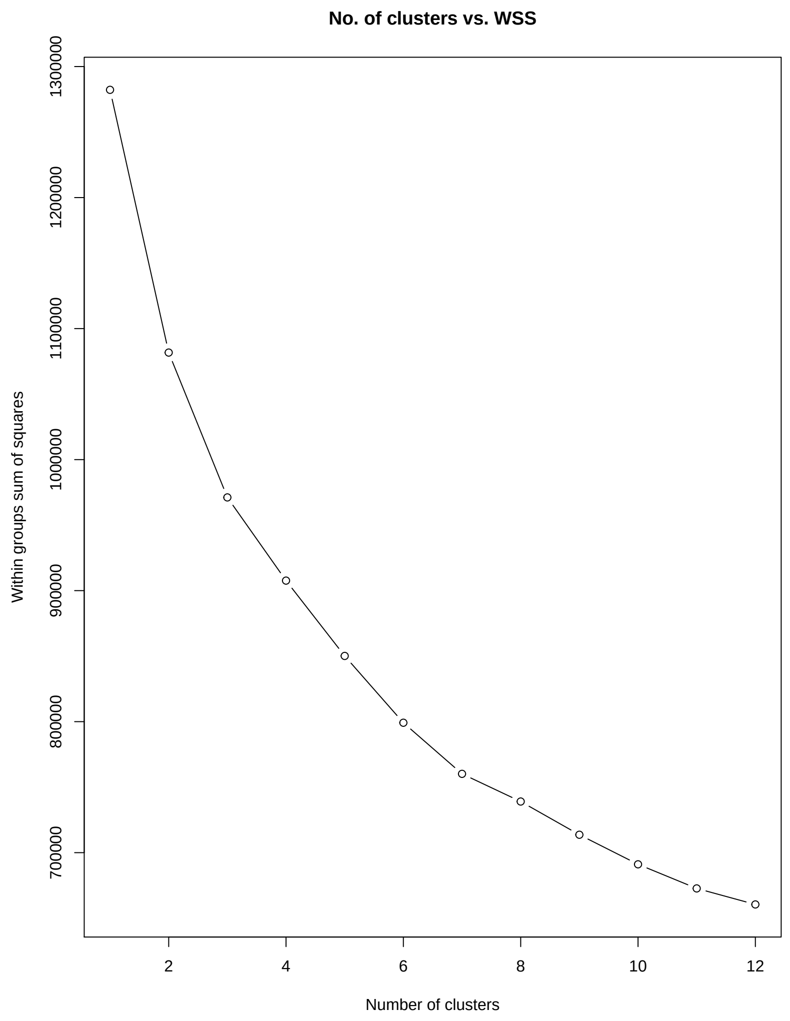

Figure 1: Decrease in SSE as k increases.

Fairly clear distinctions can be seen between the clusters. $C1$ has the highest scores on every attribute except for *buildUpPlayPassing*, indicating that the teams in this cluster are generally better than the teams in the other clusters.

Comparing clusters $C2$ and $C3$, we see that for the first 4 attributes, the teams in $C2$ tend to have higher scores than $C3$. For the next 4, the two clusters are fairly similar, with $C3$ even scoring higher on certain attributes. A deeper look into the domain-specific meaning of the attributes suggests that the teams in cluster $C2$ are better than the teams in $C3$ in matters relating to **offense**. The two clusters are on par with each other in matters relating to **defense**.

2.3 Data Preprocessing

Looking at Table 1 again, notice that Match and Team_Attributes have 115 and 25 attributes, respectively. The Match table contained the following relevant dimensions:

- season
- stage
- date
- home_team_id
- away_team_id
- home_team_goal
- away_team_goal
- goal
- shoton
- shotoff
- cross
- corner
- possession

The Team Attributes provided were the following:

- buildUpPlaySpeed
- buildUpPlayPassing
- chanceCreationPassing
- chanceCreationCrossing
- chanceCreationShooting
- defencePressure
- defenceAggression
- defenceTeamWidth

All the above listed attributes are continuous. In addition to them, the database also contained a discretized value for each attribute listed above. For the preprocessing methodology we describe below, it was necessary to choose the continuous versions of these attributes. Also, there existed an attribute called *buildUpPlayDribbling* that was not present in 969 (66%) Team_Attribute rows. This is a large percentage, and in addition to that, discarding rows with this NULL value would have had a ripple effect, causing us to discard too many matches.

Since the data was provided as a relational database, significant pre-processing steps were undertaken before modelling began.

Here, instead of tranforming the data to a new space using techniques such as Principal Component Analysis (PCA) or Multimdimensional Scaling (MDS), we have kept the data in its original state and used domain specific knowledge to hand pick some interesting features. Handpicking the features we have reduced the dimensionality complexity from 115 to a little more than 10.

2.4 Difference Vectors

During our ideation phase, we realized that when two teams are facing each other, we would need a method with which to compare the two teams. The simplest method to do this was to calculate a 'difference vector' for each match – a vector of the differences between each involved team's attributes.

The next question that arose was how to identify *which* team we were making a prediction about. Our solution was to change the problem from **Predicting the winner** to **Predicting the result for the home team**. Doing things this way fell in line nicely for our concept of 'difference vectors'. If h is a vector of the home team's attributes and a is a vector of the away team's attributes, then $d = h - a$, where d is the difference vector, *from the perspective of the home team.*

For example, see the home team h, the away team a, and their difference vector presented in table 4. The vector $d = h - a$ for each match is what is passed to our models for prediction.

	h	**a**	**d = h - a**
buildUpPlaySpeed	61	50	11
buildUpPlayPassing	46	54	-8
chanceCreationPassing	66	52	14
chanceCreationCrossing	66	54	12
chanceCreationShooting	54	46	8
defencePressure	47	36	11
defenceAggression	58	43	15
defenceTeamWidth	53	47	6

Table 4: Difference vector

2.4.1 Which attributes to use?

As mentioned in Section 2, each team had several vectors of attributes associated with it, since they had potentially differing attributes each year. As can be seen in figure 2, the distributions of team attributes were not static across the 6 years for which team information is available. As a result, for a particular team playing a particular match, we used the vector of attributes with the date *closest to the match date.* This typically meant we used attributes from the same sporting season.

2.4.2 Defining the outcome variable

Recall that, for each match, the database provided a home_goal count and an away_goal count. Since we had turned our problem into a prediction for the home team, we defined an *outcome* categorical variable for each difference vector. The variable has 3 levels: 0 if the home team lost, 1 if the match was a draw, and 2 if the home team won.

We also defined a *win_percentage* variable that looked at past encounters between two given teams. This variable will be discussed in further detail in Section 3.

3 Prediction Algorithms

We started by predicting if the home would *win* or *not win*. Hence, we changed the outcome of the match from 0 (lose), 1 (draw), and 2 (win) to just 0 (lose or draw) and 1 (win). After processing the data in a way that will be suitable for our intended task, we used various classification and regression models to come up with the outcome given two different teams and their respective attributes. The algorithms we used are presented in this section.

3.1 Decision Tree v1 – In Python

. **Attributes Used:**

- buildUpPlaySpeed
- buildUpPlayPassing
- chanceCreationPassing
- chanceCreationCrossing
- chanceCreationShooting
- defencePressure
- defenceAggression
- defenceTeamWidth

Based on what we mentioned in our proposal, we initially implemented a decision tree classifier in Python. This was done in the extremely early stages of the project, mainly as a proof of concept to learn how to implement predictive models using software.

Unfortunately, this initial decision tree did not perform particularly well. We achieved approximately 50% accuracy, which is essentially the accuracy of a random guess for a binary variable. To improve these results, we decided to try another classification method. We revisited a decision tree classifier at the end of the project, implemented in R instead of Python (section 3.5).

3.2 Binomial Logistic Regression - Predicting Win or Not Win

We went through several versions of the logistic regression, making strategic changes to the features involved each time. The details of each iteration are presented below.

3.2.1 Iteration 1

The Match table contained betting odds as published by 10 bookkeeping companies. The odds were presented as the odds of the home team winning (H), the odds of a draw (D), and the odds of the away team winning (A). We added one company's odds to our featureset (for a total of 3 additional variables). We trained a stepwise logistic regression using these features to predict 0 (lose or draw) or 1 (win).

Performing 5-fold cross validation, each time using 80% as the training set and 20% as the testing set, we achieved 65% accuracy. This was encouraging, since it represented a 15% jump from the accuracy achieved with a decision tree. However, the Odds variables represented other professionals' predictions of the match outcome. Hence, we did not want to use these predictors in our predictor. Therefore, we decided to remove those from the model.

3.2.2 Iteration 2

We re-ran the logistic regression model without the Odds variables, with the same difference vector (table 4) as we used for the decision tree. The average accuracy achieved by the logistic regression dropped to 54%. The increase in accuracy from that of a decision tree is not significant in this case.

The confusion matrix for one of the testing runs (5000 data entries) is given in table 5.

		Predicted class	
		0	1
Actual	0	2694	9
Class	1	2410	13

Table 5: Logistic Regression Confusion Matrix
Avg. Accuracy = 54%

The estimate table for the logistic model is presented in table 7.

3.2.3 Iteration 3

An accuracy of 54% was a slight improvement from the Decision Tree, but not by much. We could do better.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-0.1835	0.0142	-12.95	0.0000
buildUpPlaySpeed	0.0017	0.0011	1.57	0.1163
buildUpPlayPassing	-0.0120	0.0011	-11.15	0.0000
chanceCreationPassing	0.0035	0.0011	3.21	0.0013
chanceCreationCrossing	0.0061	0.0010	6.09	0.0000
chanceCreationShooting	0.0050	0.0011	4.57	0.0000
defencePressure	0.0131	0.0014	9.72	0.0000
defenceAggression	0.0020	0.0012	1.64	0.1016
defenceTeamWidth	-0.0067	0.0013	-4.98	0.0000

Table 6: Estimates for the 2nd logistic regression

We decided that some new meaningful feature might help boost our accuracy. To do this, we came up with a home team 'Win Percentage' attribute for each match, and we added this to our feature vector for use in model-training. We define 'Win Percentage' as follows:

> Given a match M, we find all previous encounters between the two teams involved[1]. We calculate the home team's win percentage in these previous encounters. Keep in mind that the home team in match M might not necessarily be the home team in previous matches, so we identified the 'team of interest' by team_id during these calculations.
>
> If two teams had draws in all of their previous encounters there would be a possibility of a zero win-percentage. To avoid the model thinking that this meant our 'team of interest' had *lost* all their games, we treated draws as wins for both teams.

Or, mathematically:

$$win\_percentage\_team1 = \frac{\text{matches won by team1}}{\text{matches won by team1} + \text{matches won by team2}}$$
$$win\_percentage\_team2 = \frac{\text{matches won by team2}}{\text{matches won by team1} + \text{matches won by team2}}$$

We also calculated a 'Possession Percentage' indicating the home team's possession percentage.

These calculations significantly increased pre-processing time, but the gain in model accuracy was worth it. We performed 5-fold cross-validation, and the resulting average accuracy was 74%, displaying a **20% increase** from the model without the win_percentage feature. The confusion matrix from one of the testing runs is presented in table 7.

Note: We did a stepwise logistic regression, so some of the variables may not appear in the estimate table.

The details of this model are presented in table 8.

[1]Given that the data is provided for past 6 years, we are fairly sure about a considerable number of encounters between any two teams. Hence, no matter which two teams are playing from given data, we will always come up with a non-zero win_percentage for both teams. It is highly unlikely (for this dataset) to find a combination of a teams that haven't played each other in past.

		Predicted Class	
		0	1
Actual Class	0	2451	252
	1	1149	1274

Table 7: Confusion Matrix for Logistic Regression using Win-Percentage and Possession Percentage
Avg. Accuracy = 74%

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-4.3802	0.0681	-64.34	0.0000
win_percentage	9.6293	0.1541	62.49	0.0000
buildUpPlaySpeed	-0.0029	0.0013	-2.21	0.0269
buildUpPlayPassing	0.0025	0.0014	1.83	0.0678
chanceCreationShooting	-0.0024	0.0014	-1.74	0.0823
defenceAggression	-0.0042	0.0015	-2.76	0.0057
defenceTeamWidth	-0.0041	0.0015	-2.71	0.0067

Table 8: Estimates for the Win/Not-Win Logistic Regression

3.3 Multinomial Logistic Regression - Predicting Win, Lose, or Draw

So far, we have only been predicting 0 (lose or draw) or 1 (win). The provider of the dataset mentioned that he achieved a 53% accuracy while making this prediction. We couldn't use a binomial logistic regression for this prediction, since that expects a binary outcome variable. Instead, we used a multinomial logistic regression, implemented by the R package *nnet* [1].

Predictably, average accuracy from 5-fold cross-validation went down from 74% (win or not-win) to 63% (win, lose, or draw). We also only used the win_percentage feature in this model, since the others contributed negligibly to the final equation. While not satisfactory, this achieves our mentioned goal of surpassing the author's accuracy level of 53%. The detailed model is in table 9, and the confusion matrix is in table 10.

	(Intercept)	win_percentage
1	-1.72	5.16
2	-4.72	12.79

Table 9: Coefficients for the multinomial regression

		Predicted Class		
		0	1	2
Actual Class	0	853	325	155
	1	278	375	247
	2	383	489	2021

Table 10: Confusion Matrix for Multinomial Logistic Regression
Avg. Accuracy = 63%

3.4 Ensemble Methods - Trying to do better than 63%

Implementing a multinomial regression reduced our accuracy considerably. Hence, we decided to train another stepwise binomial model that predicted 0 (win or draw) and 1 (lose). After 5-fold cross validation, we observed an accuracy of 80%. The features used by this model were 'win_percentage' and 'possesion_percentage'.

The confusion matrix for this model is given in Table 11.

		Predicted Class	
		0	1
Actual Class	0	3544	68
	1	1018	496

Table 11: Confusion Matrix for Predicting Lose or Not Lose
Avg. Accuracy = 80%

The details of the model are in Table 12.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	2.1086	0.0516	40.83	0.0000
win_percentage	-8.0385	0.1364	-58.92	0.0000
pos_percentage	0.0831	0.0252	3.30	0.0010

Table 12: Estimate Table for the Lose-Not Lose Logistic Regression

We treated these classifiers as two weak classifiers, since they are both binary classifiers and do not predict the 'Draw' outcome. We implemented a meta classifier that took into account the predictions from the Win/Not-Win and Lose/Not-Lose classifiers to return a Win/Lose/Draw prediction. Unfortunately, the accuracy came out to 53%.

The confusion Matrix is given in Table 13.

		Predicted Class		
		0	1	2
Actual Class	0	496	920	98
	1	54	981	154
	2	14	1135	1247

Table 13: Confusion Matrix from Ensemble Method

3.5 Decision Tree v2 – In R

As mentioned in section 3.1, we had initially attempted a decision tree to predict 0 (lose), 1 (draw), or 2 (win) in Python, but we ended up with a low accuracy of 50%. This time, we implemented the tree in R, but with our new and refined feature set described in the preceding sections. This newer decision tree achieved an average 5-fold accuracy of 64% – slightly better than the multinomial regression model. The tree is plotted in figure 3.

The confusion matrix is in table 14.

		Predicted Class		
		0	1	2
Actual Class	0	652	511	351
	1	145	584	460
	2	60	405	1958

Table 14: Confusion Matrix from Decision Tree
Avg. Accuracy = 64%

4 Real-World Insights

From our models, we can predict the following things:

- Win or not: We can predict with 74% accuracy whether a team will win a soccer match or not.
- Lose or not: We can predict with 80% accuracy whether a team will lose a soccer match or not.
- Match outcome: We can predict with between 63% and 64% accuracy a team's outcome for a soccer match (win, lose, or draw).

In Europe, where betting and bookkeeping are legal, there are obvious actionable benefits from such predictions. For fans world over, an empirically-evaluated prediction for a match outcome is useful and interesting information. The information can also be used for publicity purposes. During big tournaments like the world cup, for example, commentators tend to speculate on match outcomes for dramatization purposes.

From our clustering efforts, we could also see that, based on the team attributes, there are roughly three classes of teams. Coaches and players could benefit from this information, enabling them to understand teams' playing styles more deeply. It could possibly also provide more assistance in match outcome predictions.

5 Lessons Learned

This project gave us some practical insights into data mining that were not possible to get from in-class activities. For example, training and testing models in R was made much easier due to the variety of packages available with excellently written documentation. The most time-consuming part of this project was data-preprocessing. The data was provided in the form of a relational database, rather than a simple set of vectors like all our in-class examples were.

This is to be expected, since most systems collect data structured as RDBMSs in the real-world. The experience of preprocessing the data into 'difference vectors' before attempting any models was extremely useful for the following reasons:

- It helped modularize our data mining process. We extracted (or derived) the features we wanted and then attempted modelling. So when one part of the project needed fixing, the other could still function without any changes. In fact, our two processes were so separate that we pre-processed the data using Python, and trained models using R, since we preferred Python's libraries for interacting with a SQLite3 database.
- Because the pre-processing happened as a separate step (and because it took so long), it forced us to really think about *what we wanted to do with the data*. Given a dataset of this size, it is easy to get overwhelmed by all the entities (Players, Teams, Matches, etc) and begin to strike out randomly without having a plan. But our preprocessing step ensured that we *always had a plan for what we were trying to achieve* in each iteration.

We also learned was that data mining is an iterative process, much like software engineering. In section 3, we describe several iterations of our logistic model and the thought process between each iteration, and why we included or excluded certain features.

Most important of all, we got some experience 'getting our hands dirty' and implementing classifiers and predictors on real-world data, rather than 'toy' data sets.

In terms of doing things differently, we would have liked to include player attributes in our prediction algorithms, but the way the player-team relationship was presented would have made the pre-processing overly complicated. Each match had 22 attributes; 11 for the home team player ids, and 11 for the away team player ids. Querying for the average overall player rating for each team in that most relevant year would have significantly increased the complexity of preprocessing with possibly relatively little gain in model accuracy.

References

[1] William Venables Brian Ripley. Feed-forward neural networks and multinomial log-linear models. `https://cran.r-project.org/web/packages/nnet/nnet.pdf`. Accessed: 2016-12-11.

[2] Hugo Mathien. European soccer database. `https://www.kaggle.com/hugomathien/soccer`. Accessed: 2016-12-10.

[3] Ph.D Robert I. Kabacoff. Quick-r – cluster analysis. `http://www.statmethods.net/advstats/cluster.html`. Accessed: 2016-12-10.

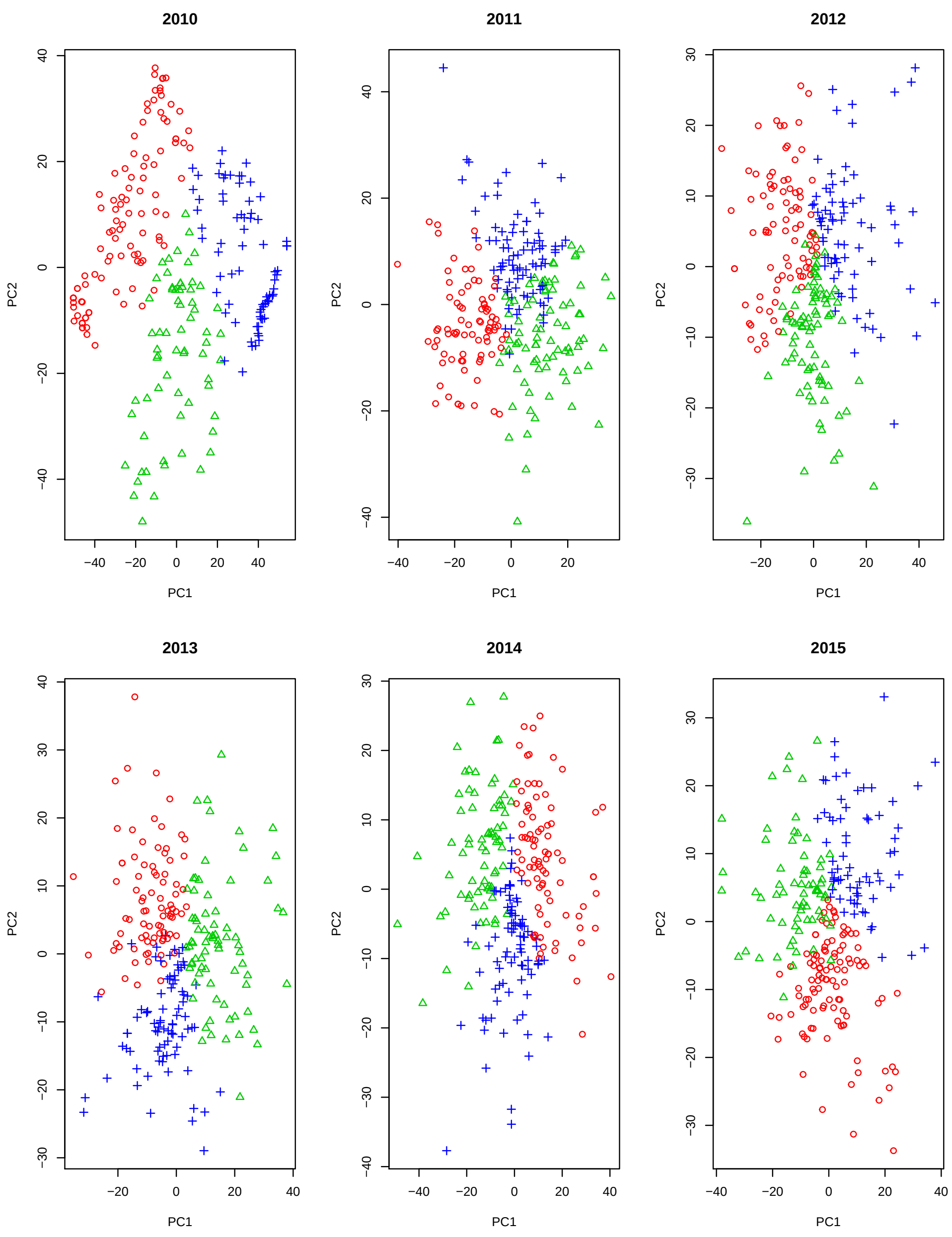

Figure 2: PCA-reduced clustered team attributes for 6 years.

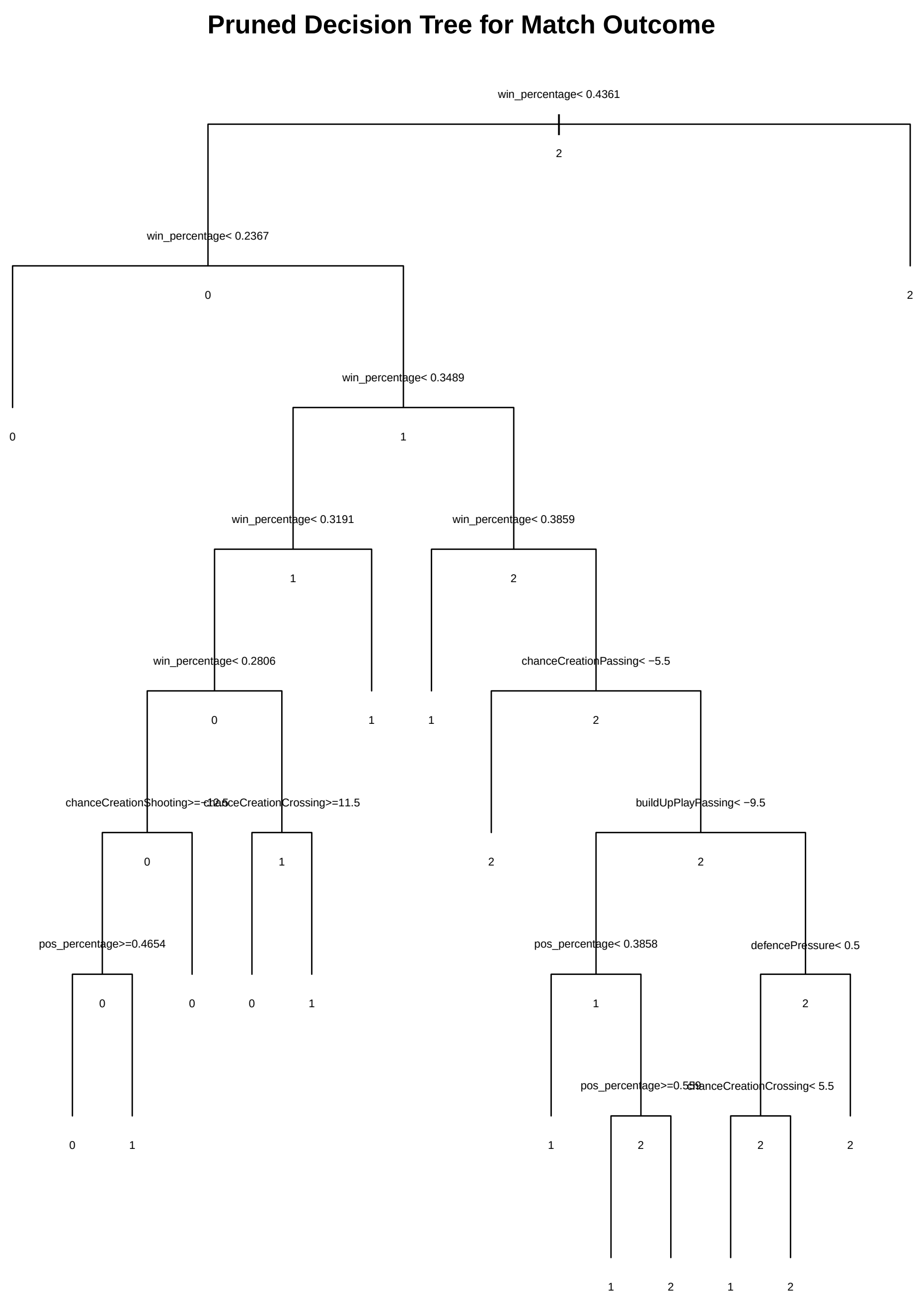

Figure 3: Decision Tree for Match Outcome