PREDICTING THE WINNING TEAM IN BASKETBALL:

A COMPLEX SYSTEMS APPROACH

LATİF CEM ÖSKEN

BOĞAZİÇİ UNIVERSITY

2023

PREDICTING THE WINNING TEAM IN BASKETBALL:

A COMPLEX SYSTEMS APPROACH

Thesis submitted to the

Institute for Graduate Studies in Social Sciences

in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

in

Management Information Systems

by

Latif Cem Ösken

Boğaziçi University

2023

DECLARATION OF ORIGINALITY

I, Latif Cem Ösken, certify that

- I am the sole author of this thesis and that I have fully acknowledged and documented in my thesis all sources of ideas and words, including digital resources, which have been produced or published by another person or institution;
- this thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- this is a true copy of the thesis approved by my advisor and thesis committee at Boğaziçi University, including final revisions required by them.

Signature…….....

Date …….....

ABSTRACT

Predicting the Winning Team in Basketball:

A Complex Systems Approach

Predicting the winner of a basketball game is a difficult task, due to the inherent complexity of team sports. All 10 players on the court interact with each other and this intricate web of relationships makes the prediction task difficult, especially if the prediction model aims to account for how different players amplify or inhibit other players. Building our approach on complex systems and prototype heuristics, we identify player types through clustering and use cluster memberships to train prediction models. We achieve a prediction accuracy of ~76% over a period of five NBA seasons and a prediction accuracy of ~71% over a season not used for model training. Our best models outperform human experts on prediction accuracy. Our research contributes to the literature by showing that player stereotypes extracted from individual statistics are a valid approach to predict game winners.

ÖZET

Basketbolda Kazanan Takımı Tahmin Etmek:
Karmaşık Sistem Yaklaşımı

Bir basketbol maçının galibini tahmin etmek, takım sporlarının doğasında var olan karmaşıklık nedeniyle zor bir iştir. Sahadaki 10 oyuncunun tümü birbiriyle etkileşime girer ve bu karmaşık ilişkiler ağı, özellikle oyuncuların diğer oyuncuları olumlu ya da olumsuz nasıl etkilediğini açıklamayı hedefleyen tahmin modelleri için, tahmini zorlaştırır. Yaklaşımımızı, karmaşık sistemler ve prototip yaklaşımı üzerine inşa ederek, kümeleme yoluyla oyuncu türlerini belirliyor ve tahmin modellerini eğitmek için küme üyeliklerini kullanıyoruz. Beş NBA sezonunda ~%76'lık bir tahmin doğruluğu; model eğitimi için kullanılmayan bir sezonda da ~%71'lik bir doğruluk elde ettik. En iyi modellerimiz, tahmin doğruluğu konusunda insan uzmanları geride bırakıyor. Araştırmamız, bireysel istatistiklerden çıkarılan oyuncu türlerinin, maç sonucunu tahmin etmek için geçerli bir yaklaşım olduğunu göstererek literatüre katkıda bulunuyor.

CURRICULUM VITAE

NAME: Latif Cem Ösken

DEGREES AWARDED

PhD in Management Information Systems, 2023, Boğaziçi University, Istanbul (GPA 4.00)
MA in Management Information Systems, 2014, Boğaziçi University, Istanbul (GPA 3.94)
BA in Business Administration, 2003, Dokuz Eylul University, Istanbul (GPA 2.74)

AREAS OF SPECIAL INTEREST

Retail Banking, Investment Banking, Capital Markets, Artificial Intelligence, Technology Risk Management, Technology Resilience, Operational Risk, Internal Audit, Public Cloud Architecture

PROFESSIONAL EXPERIENCE

Group Head of CTO and Enabling Platforms Oversight, *Lloyds Banking Group* (05.2022 - Present)
Senior Manager, *PwC UK* (06.2016 - 04.2022)
Chief Audit Execuitve, *Takasbank* (03.2013 - 06.2016)
Chief Auditor / Auditor, *Takasbank* (01.2016 - 05.2019)
Internal Auditor, *Sekerbank* (03.2006 - 12.2007)

PUBLICATIONS

International Journal Articles

Osken, C., & Onay, C. (2022). Predicting the winning team in basketball: A novel approach. Heliyon, e12189.

Osken, L. C., Onay, C., & Unal, G. (2016). Estimating defaults in organized security lending markets. *Journal of Financial Regulation and Compliance*.

Ozturan, M., Senol, D., Yilmaz, G., Osken, C., & Demirbacak, S. (2016). A Roadmap for Integrated Information Systems in Capital Markets: Case of Turkey. *International Journal of Financial Markets*, *2*(3), 94-108.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Basketball can be analysed using the complex systems theory: two teams, each with five players on the court and seven substitutes (five in the case of International Basketball Federation (FIBA) rules being used), in order to achieve a goal acting on their training and intuition whilst also reacting to each other's actions and perceived intents (García et al., 2013). One could even argue that the game is a nested complex system, with each of the two teams constituting individual complex systems, trying to win the game[1]. A similar approach, is to think of sports teams as superorganisms, with complex feedback loops and integration capabilities (Duarte et al., 2012). The complexity of such structures yields a high level of uncertainty on the likelihood of achieving the objective (Martinez, 2017) – this unpredictability is arguably a major element of what makes team sports fun! In this study, we aim to shed light on how the outcomes of basketball games can be predicted, by leveraging the dynamics of team structure.

Predicting the outcome of a basketball game is of interest to many stakeholders: coaches and team managers so they can identify optimal team compositions, betting companies so they can model the odds for financial performance, casual fans purely for entertainment purposes are but a few (Schumaker et al., 2010) of these stakeholders. Nevertheless, this prediction problem

[1] We are cognizant of the fact that in the NBA, winning games may not always be in a team's best interest. This is usually observed when a team sacrifices short term success for long-term sustained gains (i.e., better draft picks), a process also known as tanking. For the sake of parsimony, the objective statement we use excludes this approach.

is by no means a trivial one. How efficiently the components of a team interact to create synergies and adapt to opponents is not a trivial question to answer yet needs to be discussed to successfully predict game results. Therefore, we posit that understanding the players' roles and interactions is a foundational element of prediction exercises.

Historically, a strict classification of player roles has been used to help simplify the interactions within basketball teams. These roles or positions are:

- Point guard (PG) who handles the ball, dribbles up the court, sets up the offense and distributes the passes;
- Shooting guard (SG) who is apt at scoring as the name suggests, and is primarily a wing player who operates not so close to the rim;
- Small forward (SF), a versatile jack of all trades but is mainly a wing player;
- Power forward (PF), a position similar to centre, who usually stays close to the basket in basket in both offence and defence and is adept at rebounding;
- Centre (C), a position that is occupied by the tallest player on the team and operates, or at least used to, quite close to the rim.

It is worth noting that, our definition of the positions' modus operandi come with some caveats. Power forwards and centres, people who historically were confined to operate under the basket to leverage their height and physical advantages, now shoot three pointers and some even handle the ball distribution duties in their teams and have more assists per game than many point guards (due to their role as the primary distributor of the ball, PGs were historically players with highest per game assist figures by far). The game has been evolving, a la Moneyball, especially in the last two decades (Safir, 2015). In-depth statistical analysis capabilities granted by player tracking, play-by-play data, and machine learning have led the teams to seek

efficiency gains wherever they can be found, utilising synergies among certain line ups or exploiting certain weaknesses in opposing teams. This trend results in players transcending the boundaries of their positions, (Rangel et al., 2019). Standard deviation of three-point attempts per game among centres have increased from 0.2 in 2000 to 1.26 in 2017 (Figure-1), a trend that can also be seen in standard deviation of assists per game for centres (Figure-2), rebounds per game for point guards (Figure-3).

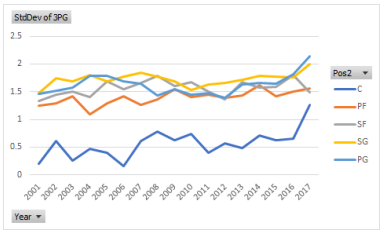


Figure 1. Standard Deviation of Average 3-Point Shots Made Per Game Per Position

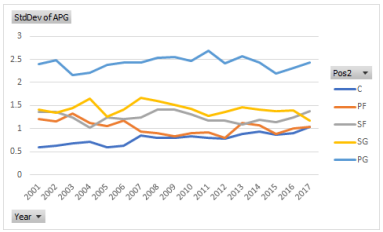


Figure 2. Standard Deviation of Average Assists Per Game Per Position

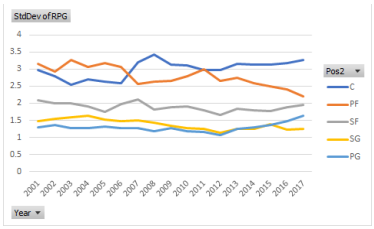


Figure 3. Standard Deviation of Average Rebounds Per Game Per Position

When the components themselves cannot be easily classified or modelled, understanding how complex systems operate become an even more difficult task. That being said, the relatively low number of agents (i.e., players) involved in sports allow us to model these environments (Passos et al., 2011) and understand how the

agents interact to enhance the probability of achieving the outcome. Therefore, we posit that the traditional player positions need to be redefined in order to capture the tendencies of player archetypes which can then be used to understand how certain archetypes yield positive or negative synergies, impacting the likelihood of team success.

1.2 Problem statement and hypotheses

The exam question our research is trying to answer is predicting which basketball team will win a game. The approach implemented uses an unsupervised learning algorithm to identify players with similar styles; followed by a supervised learning model to predict the teams winning the games using the player styles extracted in the first phase. Therein lies the novelty of our approach: building a predictor on the team configuration – i.e., how different players work together to achieve success. Therefore, we aim to analyse the hypothesis: "*The winner of a basketball game can be predicted mainly by using data on its components (i.e., players)"* using the following design principles:

i. The five traditional positions do not adequately capture player tendencies and therefore a new way of classifying players is needed; and
ii. Players occupying same traditional positions can have significantly different playing styles and therefore the correct mix of components need to be found for optimal results (e.g., guards who tend to drive frequently synergise better with centres and power forwards who are capable three-point shooters rather than low-post scorers).

As seen in the Chapters 2.1 and 2.2 on existing research on team performance and game outcomes, researchers have focused on training predictors with team level

inputs (e.g., aggregated statistics at team levels). Such aggregated statistics are unable to provide insight on the dynamics that lead to one team outperforming another (Lames & McGarry, 2007). Therefore, predictions with these approaches have limitations in their applicability. The approach presented in this thesis enables a view on how players of different styles can fare against other specific team configurations, enabling a deeper understanding of the mechanisms leading to team wins.

1.3 Structure of this thesis

Following the Introduction, Chapter 2 discusses existing research on machine learning in sports, with a specific focus on basketball, how the unsupervised (clustering) and supervised machine learning techniques we employed are applied in similar settings and problems. Building on that, Chapter 3 depicts the solution design and key principles, Chapter 4 discusses the details of the solution implementation, Chapter 5 explains the results, Chapter 6 details the inherent and discretionary limitations of this study. Conclusions are presented in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

Sports, with the inherent ambiguity they contain and the level of interest they generate, is a prime field for the application of machine learning. It is not surprising that there is a large body of research on identifying, modelling, and predicting the interactions and discrete activities related to numerous sports. In this section, we initially provide a summary of the relevant prior research on machine learning in sports by focusing on research on team performance and prediction, classification of players, and the dynamics of game and intra-game sequences.

2.1 Research on team performance and game outcomes

Kuehn (2017) uses a probabilistic model to identify how players complement or inhibit each other and models complementary skill sets. In this study, Kuehn models each possession within a game as an event tree, capturing potential outcomes. The probability of each node in the event tree is driven by another model - the individual player model. This model estimates, for every player in the line-ups of both teams, the probability for each player-event pair in the event tree. In order to estimate these probabilities, Kuehn uses player's propensities he calculates from historical play-by-play data for each player. Feeding this into the event tree, he is then calculating the expected point per possession. By doing so, this study aims to capture the interaction effects among teammates as well as the effect from the opposition's line-up when attempting to predict how well a particular line-up can fare against another. In a similar study, (Arcidiacono et al., 2017) employ a probabilistic model to estimate the

productivity spill over of players and show that team performance is significantly impacted by players' ability to help their teammates to score.

Another study that focuses on player interactions is performed by (Lutz, 2012). In his study, Lutz identifies 10 clusters of NBA players, using a combination of traditional and advanced box score statistics and a subset of shot selection data for players. He then analyses the relationship between the existence of each cluster and team success via T-tests for each cluster. Expanding on that, he also analyses the interaction effects of clusters by analysing 2 cluster and 3 cluster combinations against team success. He identifies that the "aggressive bigs" cluster has the lowest percentage of its players in winning teams, and the "perimeter scorers" cluster is usually incompatible with other clusters. We consider this study to be one of the rare studies that is trying to model the positive and negative synergies between different playing styles.

Oh et al., (2015) approach the question from yet another angle, employing graph analysis to identify the impact of player identities to team performance. Using probabilistic graphical model, they simulate the games as a series of discrete events. Nodes on their graph represent the players, players' actions, and the events; with the edges representing the likelihood of the interaction based on player's propensities. However, majority of team performance prediction studies use team-level statistics (e.g., points per game, rebounds per game aggregated at team level). (Beckler et al., 2013), for example, aggregate NBA player's game-by-game statistics at team level across '91-92 to '96-97 seasons. Using linear regression, logistic regression, support vector machines, and artificial neural networks separately, they achieve an overall 70% accuracy with linear regression. Interestingly, they find that Artificial Neural Networks (ANN) perform the worst, with only 65% overall accuracy. They also find

that all their models perform relatively poorly in some seasons and argue that these seasons are inherently more ambiguous. They compare this to other prediction sources (e.g., websites, crowd sourcing, human experts) and state that their model, in its best performing season (with an accuracy of 73%), outperforms all these sources. Cao (2012) also uses team-level inputs for NBA games, employing logistic regression, Naïve Bayes, Support Vector Machines (SVM), and ANNs separately. He achieves accuracy ranging from 66% (Naïve Bayes) to 68% (logistic regression), relatively on par with other research and human experts.

Loeffelholz et al., (2009) employ feed forward neural networks, radial basis functions neural networks, probabilistic neural networks, and generalised neural networks, fusing all 4 models through a Bayesian Model to leverage different classifiers used. Over a season's worth of games and using team-level inputs, they achieve an overall prediction accuracy of ~72%, compared favourably to human expert scores of ~69%. It is worth noting that, all but RBF NNs achieve 71 to 72% prediction accuracy, with RBF scoring 69%. In a similar study, (Miljković et al., 2010) use Bayesian classifiers with team-level inputs to predict the outcomes of '09-10 season. They report an 67% prediction accuracy, on par with human experts. (Hu & Zidek, 2004) adopt a novel approach in prediction with team-level inputs; they use weighted likelihood method to predict the outcome of the NBA Final Series of '96-97 season. They use the in-season match history of the two teams to estimate the likelihood of the teams winning each game of the finals. Their models all estimate a Bulls championship, with probabilities ranging between 61% to 71% (actual outcome was a Bulls victory). They also simulate whole season for 7 teams, using previous seasons game data but accuracy of their predictions varies significantly across the 7 teams. Obviously, their approach ignores all the roster

changes happening in the league between the two seasons as well as the player development during the summer, so these results are not surprising. In a similar study, (Orendorff & Johnson, 2007) employ Bayesian Logic Networks and Markov Logic Networks. They train their models using the outcomes of 1130 games from a total of 1230 in the ’06-07 season, then predicting the held-out 100 games. They report a 76% prediction accuracy with Markov Logic Networks and 63% for Bayesian Logic. It is worth noting that their test-training set distribution is relatively lower than the norm.

Another application of probabilistic methods, yet in college basketball setting, is the study by (Bashuk, 2009). In his study, he calculates team rankings based on cumulative win probabilities calculated by play-by-play data. He then uses the rankings to simulate the games and states that the model is able to estimate the impact of changes to a single line item in the play-by-play data, making this one of the few studies that bridge the gap between in-game data at that resolution and game outcome predictions.

Yang & Lu (2012) employ support vector machines in NBA setting. They identify the 16 playoff teams across ten seasons and analyse the games in every season between the playoff teams (i.e., 16 teams and 15 opponents for a single year yields 240 data points, this totals to 2400 across 10 years). They use SVM to predict the outcomes of the games played, using team-level inputs and achieve a 55% prediction accuracy.

One of the rare cases of applying physics to data mining in basketball, Bourbousson et al., (2010) finds that there is strong in-phase relations in player pairs in basket to basket movements as well as lateral movements. Although they operate with a small sample size of six games, their research help establish that a complex

relationship of actions and reactions exists among all the players on the field. A similar study, building on the study by Bourbousson et. al, uses control parameter as time and order parameter as the offensive rating of the teams (García et al., 2013). With this approach, the researchers analyse the periods of stability and instability and try to identify the perturbations triggering phase changes. They also identify strong in-phase relationship between teams in the first period (i.e., the first two quarters) of the game and argue that this is due to the relatively low risk appetite of teams in that period. Perturbations (e.g., timeouts, period break etc.) slowly deteriorate this stability and as the second phase goes on, anti-phase is more likely to be observed.

2.2 Research on team performance and game outcomes in other sports

Match outcome prediction, obviously, is not limited with basketball. Researchers have been trying to employ similar techniques in other settings to predict outcomes and understand the dynamics of team performances. Blaikie et al. (2011) combine team-level game statistics with data points external to the game (e.g., stadium attendance, team rankings etc.) as inputs to ANNs for National Football League (NFL) and College Football (NCAA). To test which variables, have the best predictive capabilities, they use 5 different input spaces: using only in-game statistics, using only efficiency metrics, applying data reduction methods (principal component analysis and linear regression combinatorial optimization), and using the full set of 46 input variables they gather. They analyse the results season by season and find that their models rank relatively favourably against experts for NFL but not for College Football. Another study on NFL teams uses team rankings published by The New York Times and using probit regressions estimate the outcomes of the games (Boulier, 2003). Bouiler shows that his model is able to compete with human

experts with prediction yet fail to perform better than a naïve predictor (i.e., home team wins).

David et al., (2011) have employed a committee of committees approach to merge outputs from numerous committees, each comprising 50 ANN models predicting scoring margins of NFL games. They achieve accuracy scores ranging between ~62% to ~68% across three seasons, comparing favourably to other NFL predictors.

Flitman (2006) studies Australian Football League across three seasons to create ANN models via genetic algorithms. He also assigns a probability score to outputs, showing how confident the model is in its prediction. Using this confidence score, calculates tipping scores for his model, penalizing incorrect predictions with high confidence and rewarding little for correct prediction with low confidence. This allows him, to a certain extent, to distinguish the models that have, out of luck, achieved relatively higher accuracy scores within his data set.

Leung & Joseph (2014) adopt a similar approach used Hu & Zidek (2004) but in a football context. They also use a clustering analysis to identify teams with similar playing styles and use the outcomes of games played between similar teams for prediction. In our view, this a successful way to compensate for the relatively smaller set of matches played between the same teams compared to basketball.

Min et al., (2008) adopt a novel approach to football match prediction. They employ rule-based reasoners and Bayesian networks in tandem, splitting a football match to 10 frames, each consisting of 9 minutes. The teams, based on the events and conditions pre-defined, trigger the rule-based reasoner with the strategy they can employ for the next frame and the Bayesian network takes these as inputs to simulate the frame. Outcomes of the simulation of a frame is then fed to the reasoner for the

next frame. They apply their framework to the 2002 World Cup games and report that the framework correctly identifies the 6 out of top 8 teams in the tournament, also correctly placing the top two teams in their respective positions. Another predictor is the self-organising maps, used by (Croft et al., 2015) to identify the key success criteria for rugby games.

McCabe & Trevathan (2008) use ANN with team-level inputs for rugby and football games across 4 different leagues. Their model compares favourably to numerous other tipsters (including human experts). The best tipsters in Australian Football League, Australian National League, Super Rugby, and English Premier League Football with the best respective tippers achieving accuracy scores of 68%, 67%, 75%, and 59%[2]. This study is novel in its application of coverage of different sports.

Huang & Chang (2010) and Nakhjavani et al., (2014) use ANN to predict football team's performances. Tax & Joustra (2015) also employs ANNs in football setting, alongside various other predictors and state that Naïve Bayes and ANNs perform other predictors they employ (e.g., Random Forest, decision trees etc.). Martins et al., (2017) also study football games, using polynomial classifiers as a predictor.

2.3 Research on player classification

Existing research on the field does not only focus on predicting the outcomes. A key area of focus is identifying the similarities of players. Ultimately, this is done with the aim of optimising team configurations but still these researches employ different approaches. Miller et al., (2014) use perform a spatial analysis of shot selection data

[2] Please note that football matches in Premier League can also end in draws, unlike rugby leagues. Therefore, prediction accuracy is inherently lower.

of NBA players by using spatial decomposition. They show that players have different offensive propensities in their shot selections and that the shot selection or shooting efficiencies is not always invariant to the x-coordinate of the court (i.e., symmetrical between right wing and left wing). Another study trying to identify player typologies is by Chan et al., (2012), using k-means to identify player clusters / playing styles in NHL. Hockey, unlike basketball, has less positional fluidity. Therefore, the clusters they identify represent sub-groups under goalies, defensemen, and forwards. They also analyse each clusters contribution to team performance by using linear regression and show that having a high-performer goalie has the biggest impact to team success. They also show that k-means is a potential tool for such classification exercises.

Rangel et al., (2019) assess the versatility of basketball players by using the individual box score data to predict the likelihood of that player manning each of the five traditional positions. They find that in the Brazilian Basketball League, the ratio of players not fitting into a single position is increasing rapidly, cementing our view that traditional positions in basketball are no longer adequate in capturing the play styles. (Zhang et al., 2018) employ clustering analysis to identify similar players in the NBA, agnostic of traditional positions. They also use anthropometric properties and experience. They identify five clusters yet are unable to establish any connection between team performance and the team configuration based on the clusters they identify.

Cervone et al., (2016) study the in-game value of different parts of a basketball court based on how often the teams try to occupy or own these parts, using a weighted Voronoi method. They establish that different teams have clearly different

preferences in which sections of the court they try to occupy and operate from, establishing that teams and players have different spatial tendencies.

Using a different approach to identify similar basketball players in NBA, Piette et al., (2011) adopt network modelling approach and utilise play-by-play data spanning 4 seasons. They create a network of players where nodes (players) share an edge if they have played together in a five-man line-up. The weight of the edge is identified by the offensive, defensive, and total efficiencies of the line-ups the nodes (players) shared. Fewell et al., (2012) also adopt strategic network approach to basketball and use network related measures (e.g., centrality) to classify player's roles and importance, however they lack the efficiency angle introduced by Piette et al (2011). Again in the context of NBA, Hore & Bhattacharya (2018) utilise SVM to identify player similarities but using a time series view. They use support vector machines to identify similar career trajectories of players across seasons and assess the likelihood of a given player to survive the first five years in the NBA. Dežman et al., (2001) use expert defined criteria, instead of in-game statistics, to identify the positions of a sample of basketball players from the Croatian 1st League. They find that delineating the boundaries among small forward, shooting guard, and power forward positions is not practical, further strengthening our stance on the position fluidity. Pion et al., (2018) analyse player positions in basketball from a different angle, purely focusing on physical and physiological attributes of Belgian Basketball League – First Division. They find that using these basketball-independent attributes, they are able to classify players according to their traditional positions with a very high accuracy. At first sight, that may seem as a counterargument to our point on positional fluidity, but we argue that this point further cements our thinking on these roles being defined purely on physical merits and not basketball related roles.

Another field of machine learning and statistical analysis on player performance is to produce performance rankings for players. Mertz et al., (2016) has created a ranking system to assess the success of all-time NBA players, agnostic of position or role. Vadruccio (2018) analyses player efficiency rating, wins produced, and approximate value methods used to rank or quantitatively summarise player performance. He also introduces and simulates a ranking system of his own design to rank the performance of frontcourt and backcourt players within themselves. He also recognizes the need to address positional fluidity and suggest implementation of clustering analysis to enrich further ranking endeavours.

Radovanovic et al., (2013) employ data envelopment analysis and distance-based analysis techniques to create to new metrics measuring a player's overall performance. Unlike other methods used at the time, they include player salaries into the analysis and can rank players on how successful they are compared to their cost to the team. However, majority of the research trying to either classify players or rank / compare their performance face challenges on the reliability of the inputs they use. The discussion on whether the box score (i.e., the metrics officially captured and published by the leagues) reflects the true contribution and value of a player (Martínez & Martínez, 2011), is at least two decades old. To supplement the traditional box scores, researches have created numerous advanced metrics with the hope of quantifying and summarising players' performances. Goldsberry & Weiss, (2013) and Franks et al., (2015) discuss need for additional metrics to capture defensive contribution while Maheswaran et al., (2014) shows that plain rebound statistics alone cannot capture rebounding proficiency and create additional metrics to capture box-out efficiency and positioning prowess.

CHAPTER 3

SOLUTION DESIGN

In this section, we reiterate our problem statement and present, at a high-level, the design of our solution alongside the rationale guiding the design. The technical details related to each step of the implementation is provided in the following section.

3.1 Overview

As stated in the Introduction section, our problem statement is predicting the winner of a basketball game. As seen in the literature review, to date, researchers have successfully implemented artificial neural networks and support vector machines, among other algorithms, to predict the outcomes of sports games. We opt to use ANNs and SVMs as our key prediction tools. To find a close to optimal set of hyperparameters of the ANNs we train, we employ genetic algorithms.

The novelty of our approach lies in the input space of the predictor. Majority of the research on game outcome prediction is built on either aggregating player statistics at a team level or using team level power rankings or similar proxies for performance. A handful of researchers have adopted a different path and analysed the interactions at player level, trying to capture the synergies and interaction effects among the players on the court. We agree with the latter approach due to two key main arguments:

i) The aggregation of player level statistics to team level, as argued by (Lames & McGarry, 2007) leaves much to be desired from a reliability

point of view. These measures do are usually not completely reflective of the underlying traits researchers are trying to measure (i.e., the contribution of a player to team's success), therefore forcing the predictors employed to work from an incomplete picture.

ii) In addition to our previous point, these aggregated statistics carry almost no relevant or useful information for the stakeholders involved in the analysis. Majority of the research has found that scoring more points, not turning the ball over etc. are the key statistics contributing to wins. This information has little if any practical value. A good predictor should be able to tell what a team can practically do to change the predicted outcome (e.g., changing the usual rotation of players to increase playing time for one player to exploit a weakness in the opponent).

That is why, we think understanding the interactions of the components (i.e., players) of both teams to predict the outcomes of the games is vital. We believe that the approach used by Kuehn (2017) is quite promising on that front. However, each five-player line-up, over the course of a season, faces another specific line-up only a handful of times. Therefore, this approach has limited data related to any unique 10-man (each team fielding five players at any given time) configurations. And if the players can change their propensities according to the external constraints (e.g., defenders' skills etc.), then the limited representative sample may not be enough to account for this. To mitigate this, instead of using data points for each player, we use clustering to identify players with similar playing styles within NBA and use these as inputs instead of assessing each individual as a separate data point. This prototype heuristics is widely used by our cognitive functions (usually implicitly) to enable quick analysis of complex systems (Kahneman & Frederick, 2012). Previous

research (Oberhofer & Schwinner, 2017) already show that prototype heuristics is frequently used tool in how NBA teams assess current and prospective players. Our approach leads to a trade-off: we lose precision in our input data (i.e., the difference between a player's actual skills/statistics and the skills/tendencies of the cluster representative) with the aim of increasing the size of input sample.

We also recognise the fact that players can evolve through their careers: gaining new skills and losing the athletic abilities they possessed in their youth. They can also adopt to new teammates after roster changes, further impacting their playing styles. To capture this evolution, we run our clustering algorithms once for every season. Therefore, a player can be assigned to different clusters throughout their career. For rookies (i.e., players who have recently joined the league and therefore have no historical data), we use preseason game statistics for clustering. As these statistics are usually highly inflated compared to actual games, in mid-season we rerun the clustering algorithm for rookies to re-calibre their cluster memberships. We consider this an acceptable risk as we use the playing time as a weighting mechanism for our input space. The limited playing time rookies usually get help mitigate the risk of training our prediction model with misleading data. As an additional measure, once all prediction models have been trained, we check the prediction accuracies related to each month of the season and initial months of seasons do not perform worse than the second half of the seasons (detailed results not reported).

In addition to a player's evolution over their career, we also acknowledge the fact that some players can adapt to their ecosystem and change their playing styles (Travassos et al., 2013). This phenomenon is known as phenotype plasticity in ecology literature (Werner & Peacor, 2003). This presents another potential risk in our approach: some players are highly versatile and can adapt their playing styles in

response to the configuration of their team and the opponents. We mitigate the risk of classifying these versatile players strictly within a single cluster by using two clustering algorithms in parallel: k-means and c-means. We expect that the highly versatile players that are shoehorned into a single cluster in k-means will be better reflected in the fuzzy clustering outputs and this information will improve the prediction of games involving such players.

Finally, we are also cognizant of the distorting effects of Euclidean distance in high-dimensional data sets (Aggarwal et al., 2001). Therefore, we employ, in parallel, other distance metrics in the clustering phase (i.e., Manhattan, Mahalanobis, cosine, and Chebyshev distance metrics) to account for the high dimensionality of our datasets. We select the highest performing distance metrics with respect to the Silhouette value and Calinski-Harabasz pseudo-F index values (or fuzzy partition coefficient for *c-means)* to train our predictors.

3.2 High-level design and design-related challenges

At a high-level, our solution spans three main steps: *i) data gathering, ii) clustering analysis, and iii) game outcome prediction.* Each of these steps have a few design pitfalls associated with them. Figure-4 summarises each of the three steps, related challenges, and the mitigation strategies we have adopted in our research design.

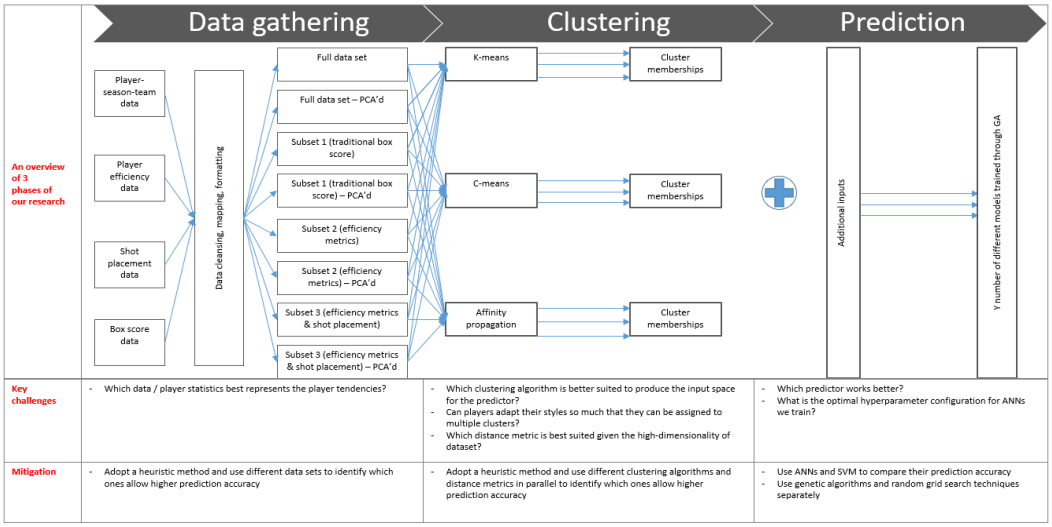


Figure 4. Challenges and Mitigation Strategies

3.2.1. Detailed view of the clustering approach

For our clustering approach, we build on the research of Lutz (2012), Miller et al. (2014) and Rangel et al. (2019). Lutz (2012) focuses on identifying players with similar styles through a data set that lacks efficiency stats (e.g., missed shots, advanced stats such as true shooting percentage etc.). Whilst Rangel et al. (2019) use traditional box score metrics to capture efficiency, the data they employ for clustering lacks the means to identify different styles, whereby Miller et al. (2014) point out in their research that the spatial distribution of players' shot selection carries significant information on the characteristics of their playing styles. We therefore employ a data set that include combinations of traditional box score statistics, advanced efficiency metrics, and shot placement data to capture both playing styles and the efficiency of each player in the cluster memberships[3].

We think it is worth to further elaborate on the challenges summarised in Figure-4. As discussed earlier, previous research has shown that tradition statistics of

[3] Please see Table-5 for the complete list of attributes used for clustering.

individual players (e.g., rebounds, points scored, assists, steals etc.), advanced statistics on player efficiency, shot placements data all carry valuable and non-trivial information. Therefore, choosing the optimal set of attributes to extract player clusters is the first hurdle. We face similar challenges in identifying the clustering algorithms and distance metrics to use as well as choosing the optimum predictor. Due to the relatively limited body of research on the field and the novelty of our approach to combine clustering and prediction methods, we are unable to answer these challenges relying on existing theory. We adopt a heuristic view and implement our approach, in parallel:

- across multiple data sets that include combinations of traditional box score statistics, advanced efficiency metrics, and shot placement data;
- multiple 3 different clustering algorithms: *k-means, c-means, and affinity propagation;*
- Euclidean, Manhattan, Mahalanobis, cosine, and Chebyshev distance metrics to account for the high-dimensionality of our datasets as suggested by (Aggarwal et al., 2001); and
- Artificial neural networks and support vector machines as predictors.

Using eight different input data sets with three different clustering algorithms, five different distance metrics, and ~25 different numbers of clusters extracted for each configuration, we perform approximately 1,240 different clustering exercises (i.e., eight data sets *x* five distance metrics *x* 25 different cluster sizes for *c-means;* eight data sets *x* 25 different cluster sizes for *k-means;* and eight data sets *x* five distance metrics for Affinity Propagation). Each of these, in theory, can be used as an input space for our predictors. That being said, some clusters perform better in identifying player characteristics than others, so we use clustering validity measures to identify

high-performing configurations. As a robustness action, we also randomly select two clustering outputs related to each dataset-clustering algorithm pair (eight datasets and two clustering algorithms totalling to 16 pairs) and use these to train predictors as well. This helps us, to a certain extent, eliminate the risk that the clustering validity measures miss certain good outcomes. We have seen that these 32 clustering outputs perform significantly worse compared to the 8 clustering outcomes selected on the basis of their validity scores and therefore do not report the prediction model outcomes for this 32 in detail. We shall discuss the prediction models' performance with the eight clustering outputs in Chapter 5.

3.2.2. Detailed view of the prediction approach

Bunker & Thabtah (2019) and Herold et al. (2019) state in their respective reviews of the existing research on machine learning in sports, artificial neural networks are one of the most widely used prediction methods. We opt to employ ANNs as our prediction algorithm, following the approaches laid out by Beckler et al. (2013), Cao (2012), Loeffelholz et al. (2009), and Yang and Lu (2012). We employ genetic algorithms to optimise the hyperparameters of the ANNs[4] we train.

Despite our earlier criticism on aggregating player statistics to team level and using these for prediction, we agree that some external events have impacts at team level and therefore need to be factored into predictive models. Therefore, we include "the number of rest days" (Bunker & Thabtah, 2019), home & away teams to capture home team advantage (Nevill & Holder, 1999), and "the calendar month in which the

[4] We also experiment with random grid search techniques but as expect these do not perform as well as the genetic algorithms so for the sake of parsimony, we only report the outputs of the genetic algorithm optimised ANNs. The results of random grid search optimisations are available upon request.

game is played" to account for any seasonality effects (e.g., tanking) (Price et al., 2010) (Soebbing & Humphreys, 2013) (Walters & Williams, 2012) to our predictors' input space. We recognise that Sampaio et al., (2010) analyse seasonality effects on player performance in Spanish Basketball League and observe no significant effects of season period. However, there is significant evidence on NBA teams performing differently in different parts of the season. Weak teams usually begin losing more frequently than they would normally be expected to, once they lose their hopes of going to the playoffs (Price et al., 2010) (Soebbing & Humphreys, 2013) (Walters & Williams, 2012). This is done with the aim of securing a more advantageous position in the next NBA draft and is also known as tanking. Therefore, we include these temporal variables in the input space of our predictors. In Chapter 4, we explain, in detail, the implementation of our solution to answer the research question and mitigate the said challenges.

CHAPTER 4

SOLUTION IMPLEMENTATION

4.1 Data gathering

The data we utilise for our research is acquired from public sources. We use seasonal player statistics for NBA players and box score data sourced from www.basketballreference.com from ’12-13 to ’17-18 seasons. For the same period, we gathered shot selection data for each NBA player from www.nbaminer.com website. We use regular season data and leave out playoff games to ensure a balanced data set for all teams.

Following a similar method to (Oberhofer & Schwinner, 2017), we removed with less than six minutes in per game per season (e.g., if a player averaged five minutes during ’14-15 season but averaged 10 minutes per game in ’15-16, he was removed from the dataset for ’14-15 but not for ’15-16), leaving us with 2,552 player-season pairs to use for clustering, and 6,150 games across five seasons. The reason we excluded the 141 data points (corresponding to c. 5% of the overall data population) from our analysis was to account for garbage time: when there is a significantly large points differential between the teams with limited time left on the clock. In these situations, when the outcome of the game is all but settled, teams venture outside their usual rotations, resting their main players and giving the fringe players the chance to gain experience. These parts of the game, while can be fast paced and yield bountiful statistics for some players, do not impact the game’s outcome and run the risk of adversely affecting our model training.

4.2 Data structure

The data we gather on player performance includes three key categories:

- Traditional basketball box score data, including but not limited with, points scored per game, rebounds per game, assists per game etc.;
- Advanced efficiency metrics, including true shooting percentage, effective field goal percentage, rebound rate, assist rate etc.; and
- Spatial distribution of player's shots across the 14 sections of the court (e.g., restricted area, in the paint but non-restricted area, right corner 3-point area, left corner 3-point area etc.).

A total of 49 attributes for each player has been collected to be used for clustering purposes. However, there is no clearly established literature on which of these attributes should be used to efficiently discriminate the different playing styles of players. Therefore, taking a conservative approach, we create 4 different datasets and use each of these separately during the clustering. A complete list of our clustering datasets, attributes for clustering and the definitions of each attribute is depicted in Table-1. Needless to say, all of these attributes were part of the input set "Full dataset".

Table 1. Player Attributes Used for Clustering

Abbreviation	Attribute definition	Included in Subset 1	Included in Subset 2	Included in Subset 3
MP	Minutes Played Per Game	☑	☑	☒
GS%	Percentage of games player was part of the starting five	☑	☑	☒
PS/G	Points scored per game	☑	☒	☒
FG	Field Goals Per Game	☑	☒	☒
FGA	Field Goal Attempts Per Game	☑	☒	☒
FG%	Field Goal Percentage	☑	☒	☒
2P	2-Point Field Goals Per Game	☑	☒	☒
2PA	2-Point Field Goal Attempts Per Game	☑	☒	☒
2P%	FG% on 2-Pt FGAs.	☑	☒	☒
3P	3-Point Field Goals Per Game	☑	☒	☒
3PA	3-Point Field Goal Attempts Per Game	☑	☒	☒
3P%	FG% on 3-Pt FGAs.	☑	☒	☒

FT	Free Throws Per Game	☑	☒	☒
FTA	Free Throw Attempts Per Game	☑	☒	☒
FT%	Free Throw Percentage	☑	☒	☒
ORB	Offensive Rebounds Per Game	☑	☒	☒
DRB	Defensive Rebounds Per Game	☑	☒	☒
TRB	Total Rebounds Per Game	☑	☒	☒
AST	Assists Per Game	☑	☒	☒
STL	Steals Per Game	☑	☒	☒
BLK	Blocks Per Game	☑	☒	☒
TOV	Turnovers Per Game	☑	☒	☒
PER	Player Efficiency Rating. A measure of per-minute production standardized such that the league average is 15.	☒	☑	☑
TS%	True Shooting Percentage. A measure of shooting efficiency that takes into account 2-point field goals, 3-point field goals, and free throws.	☒	☑	☑
eFG%	Effective Field Goal Percentage. This statistic adjusts for the fact that a 3-point field goal is worth one more point than a 2-point field goal.	☒	☑	☑
3PAr	3-Point Attempt Rate. Percentage of FG Attempts from 3-Point Range	☒	☑	☑
FTr	Free Throw Attempt Rate. Number of FT Attempts Per FG Attempt	☒	☑	☑
ORB%	Offensive Rebound Percentage. An estimate of the percentage of available offensive rebounds a player grabbed while he was on the floor.	☒	☑	☑
DRB%	Defensive Rebound Percentage. An estimate of the percentage of available defensive rebounds a player grabbed while he was on the floor.	☒	☑	☑
TRB%	Total Rebound Percentage. An estimate of the percentage of available rebounds a player grabbed while he was on the floor.	☒	☑	☑
AST%	Assist Percentage. An estimate of the percentage of teammate field goals a player assisted while he was on the floor.	☒	☑	☑
USG%	Usage Percentage. An estimate of the percentage of team plays used by a player while he was on the floor.	☒	☑	☑
STL%	Steal Percentage. An estimate of the percentage of opponent possessions that end with a steal by the player while he was on the floor.	☒	☑	☑
BLK%	Block Percentage. An estimate of the percentage of opponent two-point field goal attempts blocked by the player while he was on the floor.	☒	☑	☑
TOV%	Turnover Percentage. An estimate of turnovers committed per 100 plays.	☒	☑	☑
Above the Break 3-Usage	Percentage of player's shots taken from the 3-point area above the break	☒	☒	☑
Mid-Range Usage	Percentage of player's shots taken from the point zone but outside the paint	☒	☒	☑
In The Paint (Non-RA)- Usage	Percentage of player's shots taken from within the paint but outside of RA	☒	☒	☑
Restricted Area-Usage	Percentage of player's shots taken from within the RA (including dunks and layups)	☒	☒	☑
Right Corner 3-Usage	Percentage of player's shots taken from the 3-point area in the right corner	☒	☒	☑
Left Corner 3-Usage	Percentage of player's shots taken from the 3-point area in the left corner	☒	☒	☑
Backcourt- Usage	Percentage of player's shots taken from the backcourt	☒	☒	☑
Above the Break 3 %	Making percentage of player's shots from that specific area	☒	☒	☑
Mid-Range %	Making percentage of player's shots from that specific area	☒	☒	☑
In The Paint (Non-RA) %	Making percentage of player's shots from that specific area	☒	☒	☑
Restricted Area %	Making percentage of player's shots from that specific area	☒	☒	☑
Right Corner 3 %	Making percentage of player's shots from that specific area	☒	☒	☑
Left Corner 3 %	Making percentage of player's shots from that specific area	☒	☒	☑
Backcourt %	Making percentage of player's shots from that specific area	☒	☒	☑

We then normalise our data to mitigate the nominal attributes (e.g., points per game, rebounds per game) dominating ratio-based attributes (e.g., free throw percentage). For each dataset used in clustering (hereafter referred to as Full dataset, Subset 1 for traditional box score metrics, Subset 2 for advanced efficiency metrics, and Subset 3 for advanced efficiency metrics and shot placement data), we also apply dimension reduction to mitigate the hyper dimensionality effects as identified by Aggarwal (2001). We apply Principal Component Analysis to each of the four datasets with cut offs defined at 90% of the cumulative variance explained and identify, respectively 15, 6, 7, and 15 components. We use the PCA outputs, as well as the four initial datasets during the clustering phase, effectively running each clustering configuration 8 times for each of these inputs sets.

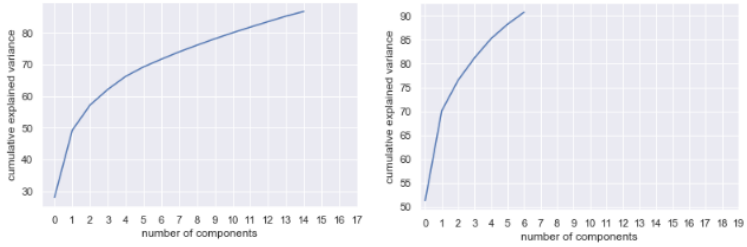


Figure 5. Number of Components Extracted Vs Cumulative Variance Explained for Full Dataset (Left Hand Side) and Subset 1 (Right Hand Side)

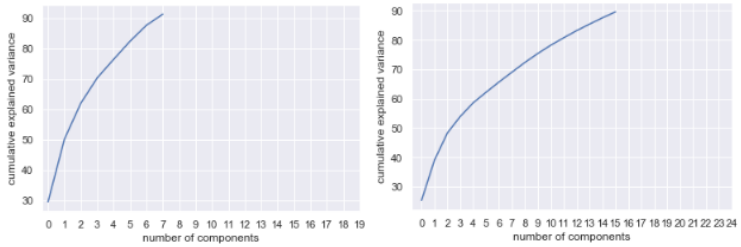


Figure 6. Number ff Components Extracted Vs Cumulative Variance Explained for Subset 2 (Left Hand Side) and Subset 3 (Right Hand Side)

4.3 Clustering

We briefly introduced our design, incorporating 3 different clustering algorithms separately, in Section 3. In this section, we shall explain the implementation and parameter optimisation for these clustering algorithms.

4.3.1. Clustering with the k-means algorithm

The first clustering algorithm we implemented was k-means, with Euclidean distance as the distance metric, using the Python libraries by (Pedregosa et al., 2015). The algorithm, for each value of *k* (i.e., number of clusters to be extracted), was run 20 times with different centroid seeds, with the best output with regards to inertia being selected as the output (please see Appendix A for the implementation code for *k-means, c-means, and affinity propagation)*. Having previously established that the traditional five positions are not fit for purpose to capture and explain the playing styles of modern players, we are left with a mini quest to identify the best *k* for our clustering exercise. Instead of committing to a small set of potential values for *k* and then assessing their validity retrospectively, we adopt a style more akin to brute force. We run our k-means algorithm for all values of *k* between five and 35. For each of these clustering outputs, we estimate the cluster validity using Silhouette value and Calinski-Harabasz pseudo-F index for each run of the algorithm (Chan et al., 2012). We store these values in a list each time we run the k-means algorithm for each of the eight input sets, for a total of 240 times. Using these two validity indices, we identify best performing *k* values by ranking the model outputs from best to worst with respect to each index separately. We see that the top 10 performers for both indices overlap. We find that *k* values of 25, 27, 26, and 23 yield the highest clustering validity scores respectively for the PCA applied versions of full data set,

subset 1, subset 2, and subset 3. In the following section, we shall discuss in detail, the members of k=26 for Subset 1 and from a domain knowledge point of view their validity, as well as assigning labels to each cluster. The key objective of our research is predicting game outcomes and while it is critical that the inputs (i.e., cluster memberships) to the predictors are reliable and valid, this is not the focus of our research so we shall limit the details of cluster memberships to a few select examples.

It is worth noting that, at this time, none of these clustering outputs necessarily constitute a good input space for our predictors. Bezdek (1981) state that clustering is an imperfect tool and no validity measure is able to capture the correct number of underlying clusters in every case. Armed with this ominous warning, we also use randomly selected sample from the non-selected *k* values to use as inputs for our predictors. All of these randomly selected inputs perform significantly poorer than the ones selected based on the validity indices. We repeat this preventive measure for each of the remaining two clustering algorithms too, with same results (we do not report the detailed results of these predictive models trained using randomly selected clustering outputs).

4.3.2. Clustering with the *c*-means algorithm

In order to account for the phenotype plasticity players can exhibit, we also include a fuzzy clustering method in our toolkit. Being able to identify a player's propensities across numerous playing styles is a valuable capability to differentiate versatile players and model their contribution to team performance across all these various styles. We use the Python libraries created by (Virtanen et al., 2020) to implement

the c-means clustering. Our approach to implementation is virtually identical to our approach for k-means, with two key differences:

- In addition to Euclidean distances, we also use cosine, Manhattan, Mahalanobis, and Chebyshev distance metrics for our c-means implementation. Aggarwal et. al (2001) prove that high dimensional data sets have some counter-intuitive properties related to similarity measures. Euclidean distance-based similarity measures become ill-defined as the distances between a target point's nearest neighbours and farthest neighbours tend to converge.
- Instead of the Silhouette value and Calinski-Harabasz pseudo-F index to estimate clustering validity (as we did for k-means), we use the partitioning coefficient, a validity score built on the distribution of cluster memberships (Bezdek & Bezdek, 1981). (Wang & Zhang, 2007) show that membership-based validity indices can perform as well as the validity indices using the membership and the underlying data in tandem. For sake of parsimony, we think partitioning coefficient by itself suffices to estimate clustering validity. Although, we also note that another school of thought on this issue argue that in the absence of pre-defined membership, cluster validity cannot be effectively established (Ross, 2010).

We also think it is worth stressing that we use the probabilities of being assigned to a cluster as a proxy for the time a player allocates to the playing style the cluster represents. And we further extend this assumption such that we assume the distribution of time across different clusters a player is assigned is invariant to the game or the opponent (e.g., if a player has a .4 for cluster A and .6 score for cluster C, we assume he spends 40% of his minutes playing with the style cluster A

represents, with the remainder of his minutes dedicated to cluster C). We recognise that this is highly unlikely to hold true and the players would fully try to utilise their skillset to eke out any marginal advantage they may posses. In the previous example, if the opponent is particularly vulnerable to cluster C, our player would likely spend much more of his minutes in that style. However, we have not designed a way to model such changes in a game-by-game resolution. We also consider the law of marginal returns and the fact that the opponents would likely change their strategy to minimise any such exploits, so we consider our assumptions to be reasonable.

Our clustering outputs show that cosine distance consistently outperform other distance metrics by a large margin in our datasets. Based on the validity score we employ, full data set and subset 1 with 10 clusters and again subset 1 with 12 clusters are the best performing c-means configurations, by a large margin.

4.3.3. Clustering with the Affinity Propagation algorithm

Both k-means and c-means clustering algorithms share the same vulnerability: their outputs are quite sensitive to the input parameter *k:* value of clusters to be extracted. We have shown in our literature review and solution design sections that currently, a clear set of playing styles has not been established (Rangel et al., 2019; Zhang et al., 2019). Therefore, both algorithms are heavily impacted by the value of *k* we define. Even though we use a reasonably wide range of *k* values to mitigate this shortcoming, we cannot be certain that the optimal value actually lies within this range. Therefore, we use a third clustering method, affinity propagation, again using the Python libraries by (Pedregosa et al., 2015). Frey & Dueck (2007) and later Dueck (2009) introduce this new clustering algorithm founded upon the principle of message exchanges between each data point in the clustering set to identify

similarities. The elegance of their method is that the algorithm identifies the number of clusters based on the data so no prior assumptions on this parameter are required. The only inputs required for affinity propagation are the similarity matrix between the data points, and the preference scores (i.e., the control knob identifying how likely a certain data point will be assigned as a cluster centroid). The preference scores also affect the number of clusters to be extracted.

To compute the similarity matrices, we opt to choose a similar set of distance metrics we employed for c-means, the only difference being squared Euclidean instead of Euclidean as recommended by (Frey & Dueck, 2007).

We also experiment with the three approaches Frey & Dueck (2007) recommend setting the preference scores and find that using a shared preference score for each data point (as we don't possess a priori knowledge of which) and setting the preference scores of each item to the median of the similarity matrix (to get a relatively moderate number of clusters to be extracted) yield the best performing clustering outputs with regards to prediction accuracy. We also employ Silhouette value and Calinski-Harabasz pseudo-F index to estimate clustering validity for affinity propagation as well, using the same methods with our k-means implementation. Using these validity estimators, we select the best performing distance metrics: subset 1 with squared Euclidean distance (with 101 clusters extracted) and cosine distance (with 80 clusters extracted) to be used as input spaces for our predictors.

4.4 Prediction

Nevill & Holder (1999) have studied the existing research on home team advantage and the potential reasons underpinning the phenomenon. To reflect this advantage, we construct our input space such that one class of outcomes always represent home team wins and the other class always represent away team wins.

To create the input space for predictors, we map the cluster memberships generated to team rosters, on a game-by-game basis. Whilst doing so, we use the minutes played for each player as a weighting mechanism in order to account for the time they are on the court. As stated before, we also include the following team-level attributes that are external to the game:

- Number of days between the game date and the last game played by home and away teams;
- Winning percentage of each team up to the game date in that particular season; and
- Dummy variables for the month of the season.

Table – 2 below depicts the input space for the Orlando Magic @ Indiana Pacers game on 29/10/2013, including the clustering outputs and the team level attributes fed to the ANN.

Table 2. Example Input Space for Predictors

Input Attribute	Value	Input Attribute	Value
home team rest day	0	away20	0
away team rest day	0	away21	0.058091
Home WP[5]	N/A	away22	0
Away WP[6]	N/A	away23	0.128631
October	1	away24	0
November	0	home0	0
December	0	home1	0
January	0	home2	0
February	0	home3	0
March	0	home4	0.062241
April	0	home5	0.120332
away0	0	home6	0
away1	0	home7	0
away2	0.107884	home8	0
away3	0	home9	0.149378
away4	0.128631	home10	0.149378
away5	0.26556	home11	0
away6	0	home12	0.037344
away7	0	home13	0
away8	0	home14	0
away9	0	home15	0
away10	0	home16	0
away11	0.037344	home17	0
away12	0	home18	0
away13	0.095436	home19	0.153527
away14	0.078838	home20	0.240664
away15	0	home21	0.087137
away16	0.099585	home22	0
away17	0	home23	0
away18	0	home24	0
away19	0		

[5] Please note that due to this game being the first game of the season for both teams, this data point is null.

[6] Please note that due to this game being the first game of the season for both teams, this data point is null.

4.4.1. Artificial neural networks

As Bunker & Thabtah (2019) and Herold et al., (2019) state in their respective reviews of the existing research on machine learning in sports that artificial neural networks are one of the most widely – if not the most widely – used prediction methods. Mimicking the biological non-linear neural networks defined by Grossberg, (1988), artificial neural networks are powerful tools to identify the complex – and usually non-linear – relationships between the inputs and the outputs. As we stated in the introduction section, the outcome between the likelihood of achieving the objective for a team (i.e., winning the game) and the components available is a complex dynamic. Therefore, we use ANNs as our primary predictors. We also employ support vector machines as an alternative predictor and shall explain this implementation and differences between the cost-efficiency of these two methods later.

We use a 75%-25% random training-test split, totalling to 4,612 game scores for training 1,538 game scores for testing for our predictor training. We deliberately refrain from selecting a single season as test data in order to avoid any seasonal or longer-term temporal idiosyncrasies impacting our models.

We opted to use multi-layer perceptrons (MLP) for our research as, by their nature, MLP lend themselves quite well to the problem at hand, with their ability to discriminate among many input attributes by the weights associated with each and extracting or emulating the complex relationships through their hidden layers. Yet, the problem of finding the optimal hyperparameters for ANNs also apply to MLP. In order to find the optimal model configuration for our predictors, we employ genetic algorithms.

Genetic algorithms, compared to random searches, are more efficient and eloquent solution to the hyperparameter optimisation problem. By exploiting the relatively higher value portions of the solution space through increased representation of these regions via reproduction over generations, these algorithms are an efficient way to address this issue (Holland, 1992) (Belew et al., 1992).

Flitman (2006), Tax & Joustra (2015), and Nakhjavani et al., (2014) have all successfully employed genetic algorithms to improve their prediction models in sports analytics.

We have built our implementation of genetic algorithm ANNs (GA ANN) by modifying the DeepEvolve framework (Liphardt et al., 2019). The ANNs are built on TensorFlow using Keras (please see Appendices B to H for the Python code for the implementation), with the genome of each model coding data on:

- Number of layers between 2 and 5 (inclusive);
- Number of neurons in each layer randomly selected from a list of (4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 256) neurons;
- Activation function is selected randomly from within the following list (rectified linear unit function, exponential linear unit function, softplus, softmax, sigmoid activation function, hard sigmoid activation function, hyperbolic tangent activation, and identity function);
- Optimization function is selected randomly from within the following list (RMSProp optimizer, Adaptive moment estimator, Stochastic gradient descent optimizer, Adagrad optimizer, Adadelta optimizer, Adamax optimizer (Kingma & Ba, 2015), Nesterov Adam optimizer; and
- Learning rate is randomly selected from within the following list (0.1, 0.01, 0.001, 0.0001).

It is worth noting that we have not included regularization term within our genome. We have opted to use validation accuracy as the measure of fitness at the end of each epoch in the implementation of our genetic algorithms. Therefore, model training and early stopping is driven by test accuracy, limiting the risk of overfitting the model to the training data to a sufficiently negligible level. For each input version (i.e., each clustering output we used to train predictors), we train 300 generations, beginning with 500 individuals in a generation (for the 8 high performing clustering outputs plus the randomly selected 16 outputs, this totals to 3,600,000 ANN models trained using genetic algorithms). For each generation. we retain the fittest 25% of that generation, as well as randomly selecting 10% of the rest of the individuals. Following breeding, we employ a 10% probability of for each gene to undergo a random mutation. Finally, we cross check the new individual's genome with the list of historically trained models (by comparing genome hashes) to avoid introducing a clone of a surviving or dead model.

Our input space, especially for the outputs generated with relatively large values of *k (*i.e., number of clusters) will obviously yield relatively sparse inputs. Whilst sparsity presents certain challenges for neural networks, they can also present benefits on efficient disentangling of information due to usually loosely coupled inputs in the sparse data and also a higher likelihood of being linearly separable (Glorot et al., 2011). We include a mix of linear and non-linear activation functions in our genome to exploit this benefit.

As expected, genetic algorithms outperform SVMs in terms of prediction accuracy but at the cost of computing time. We shall analyse the accuracy and time of these two versions in detail in Chapter – 5, Analysis.

4.4.2. Support Vector Machines

Support vector machine (a.k.a. support vector network) is a highly efficient classifier for two-group problems, introduced by (Cortes & Vapnik, 1995). Mapping the input vectors non-linearly to a high-dimension feature space, SVM constructs a decision surface delineating the classes. The high-level of generalization performance achieved by SVM is invariant on the dimensionality of the input space (Cherkassky & Ma, 2004), making SVM an excellent alternative to ANNs for our research, especially for the relatively higher dimensionality inputs.

We implement SVM, again using the Python libraries by (Pedregosa et al., 2015) (please see Appendix I for the Python code used in the implementation). For each input set, we run linear, sigmoid, and polynomial kernels separately, with the regularization parameter set at 1, degree of polynomial set at "3" (only applicable to polynomial kernel), gamma values set at 1/n where n is the dimensionality of the input space (we also tested gamma values of 0.0001, 0.001, 0.1, 0.5 and saw that linear kernel was relatively insensitive to gamma values with our datasets while polynomial and sigmoid kernels saw relatively small boosts in prediction accuracy). Across our eight input sets, linear kernel consistently outperformed other two kernels.

CHAPTER 5

ANALYSIS

In this chapter, we discuss the results of our analysis in two phases, starting with the clustering performance and the interpretation of a sample of the clustering outputs; followed by a discussion on the performance of our prediction models in detail. As stated earlier, we assess the validity of the clusters extracted through clustering validity indices. Whilst these do not perfectly capture how well the clusters have been performed, we feel confident that these can be relied upon as an interim measure to identify the set of clustering outputs most likely to train high-accuracy predictors.

5.1 Clustering analysis

Using Silhouette Score and Calinski-Harabasz pseudo-F index, we analyse the genesis of k-means and Affinity Propagation algorithms. For k-means we see that the input set “Subset 1 with PCA applied”, with 24 to 28 clusters to be extracted, is the highest performer across both validity scores. “Subset 2 with PCA applied”, with 25 to 28 clusters to be extracted follows. Best performing clustering configuration for “Subset 3, normalised” are 24, 25, and 26 clusters but they perform slightly poorer compared to the first two datasets. As for our “Full dataset”, again the highest performing clustering outputs are achieved by 25 and 26 clusters, with PCA applied. Our k-means results clearly point to an optimal range for *k,* between 24 to 28. It is worth noting that, as stated earlier, we have only used Euclidean distances during our implementation of k-means algorithm.

Our c-means algorithms yield starkly different products. “Subset 1 with PCA applied” and “Subset 1 – normalised” are the best performing input but with just 5 to 12 clusters using cosine distances. “Subset 2 with PCA applied” and “Subset 2 – normalised” follow suit, with relatively worse PC scores. Existing research shows that in clustering with high-dimensional data, cosine distance can outperform Euclidean distance (Sahu & Mohan, 2015). Using cosine metric generated outputs lead to an interesting trade off: players with similar tendencies (e.g., ratio of shots taken across each region, assist/rebound/points ratios etc.) will be more likely clustered even though their nominal output may be different. As an overly simplified example:

- Player A has 20 points, 8 assists, 8 rebounds per game;
- Player B has 10 points, 4 assists, 4 rebounds per game; and
- Player C has 19 points, 9 assists, 1 rebound per game.

In this example, Euclidean distance would tell us that Player A is more similar to Player C whereas cosine distance would yield a greater similarity between Player A and Player B. With cosine distance metric sacrificing the difference between quantities per attribute for the sake of identifying vectoral similarities, we feel that clusters/prop that may be overlooked by k-means could be captured.

The other distance metrics (i.e., Chebyshev, Manhattan, Mahalanobis, and Euclidean) perform significantly worse in our c-means implementation, across all ranges of cluster numbers.

Affinity propagation, across all our input sets, perform significantly poorer (in terms of Silhouette Score and Calinski-Harabasz pseudo-F index) relative to k-means. Best performing input set with this algorithm is “Subset 1 with PCA applied” with highest validity score achieved by squared Euclidean distance metric (yielding

101 clusters), followed by cosine distance metric (yielding 80 clusters). Remaining input sets show a significantly poorer performance on validity scores.

We opted to discuss, in detail, one of the highest performing clusters, Subset 2 with PCA applied, clustering algorithm as k-means and *k* value of 26. This configuration has performed strongly not only in validity indices but also in training predictors with high accuracy (~76%, we shall discuss prediction accuracy of our models in Chapter 5.2 – Game outcome predictions*)*. We want to emphasise the fact that the cluster structure we shall dissect in this section is but one of the high performing intermediate products. It is the one that has led to the training of the predicting model with one of the highest accuracy levels we achieved but by no means this guarantees that it is the best clustering structure explaining the dynamics and player roles. In Table – 3 below, we discuss the 20 most prominent clusters (out of 26 clusters[7]) in terms of minutes played, usage rates, and their impacts on key statistical attributes alongside examples of prominent members for each cluster. Please note that whilst each cluster is labelled during our analysis, some inevitably contain fringe players in terms of minutes played and statistical contributions, so we only focus on the more interesting clusters in this example.

[7] The cluster labels not explained in the table are: 'passer-snipers', 'point forwards', '3&D', 'fringe bigs', 'mid-tier scorers', and '3-point specialists'.

Table 3. In-Depth Analysis of A Clustering Output Example

Cluster Label	Comments and characteristics	Some of the notable members
Elite bigs	This cluster, as the name suggests, is comprised of efficient and multi-faceted players who can rebound, assist, and score in the paint and midrange quite efficiently, despite their incredibly high usage rates.	’14-15, ’15-16, ’16-17, and ’17-18 seasons of Anthony Davis ’13-14 and ’14-15 seasons of Tim Duncan
Offensive juggernauts	These players have no offensive weak spots, despite their high usage rate, they distribute their shots relatively well among midrange, painted area, and 3 pointers. They also boast healthy assist rates, placing them as the focal point of their team’s offence.	’14-15 and ’15-16 seasons of Kobe Bryant ’13-14, ’14-15, and ’15-16 seasons of Kyrie Irving
Swiss army knives	These players are the embodiment of versatility. Their shot selection is similar to “All around scorers”, albeit with less midrange usages and more 3 pointers. Yet they contribute more in rebounds, assists, steals, and blocks, leading to a high efficiency rating.	’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Kevin Durant ’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of LeBron James ’16-17 season for John Wall
Bruisers	These players live and die in the restricted area. They have incredibly high rebounding and block rates, take an astounding of 74% of their shots from the restricted area. They also boast very high player efficiency ratings and shooting efficiency.	’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Dwight Howard ’14-15, ’15-16, and ’16-17 seasons of Hassan Whiteside
Offensive focal point	These players have a unique blend of high usage rate, assist rates, and painted area usage. They prefer to shoot relatively less 3 pointers compared to similar groups with similar assist and / or usage statistics. They also have low rebound rates and block rates, further suggesting that their key duty is setting up their teammates to score and scoring predominantly via lay ups or post up plays.	’13-14, ’14-15 (for both Boston Celtics and Dallas Mavericks), ’15-16 (both for Sacramento Kings and Chicago Bulls), ’16-17, and ’17-18 seasons of Rajon Rondo ’13-14, ’14-15, ’15-16, and ’17-18 seasons for John Wall
Jack of all trades	This cluster is a slightly different version of the Swiss Army Knives, they still boast large usage rates, efficiency figures across the board but with lower 3 pointers and assist rates and higher rebound and painted area usage.	’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Dirk Nowitzki ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Al Horford
Combo wings	Combining a high usage rate and a large portion of shots from the 3-point territory, these players also have a relatively high level of assist rates. This suggests that they span the traditional point guard and shooting guard duties as their natural modus operandi.	’13-14, ’14-15, ’15-16, and ’17-18 seasons of Manu Ginobili ’13-14, ’14-15, ’15-16, and ’16-17 seasons of J.J. Barea
Stretch bigs	These players take more than 50% of their shots from the 3-point territory and yet still manage a 25% restricted area usage. They deliberately prioritise high-value shots, have relatively high rebounding rates, efficient shooting, and low assist rates. They are the front court players of the new era, foregoing low value post up play for the sake of high value 3 pointers and space created for their teammates.	’14-15 season of Ryan Anderson ’14-15, ’15-16, and ’16-17 seasons of Nikola Mirotic
Slashers	These players make their offensive impact from inside the 3-point line, shooting more than 70% of their shots from midrange or restricted area. Despite their relatively lower shooting efficiency compared to classes discussed above, they maintain a high usage rate and a healthy assist rate.	’13-14 and 15’16 seasons of Kyle Lowry ’15-16 season of Dwayne Wade
Playmakers	These players display a fair distribution of their shots across the 3-point area, midrange, and painted area. They also have high usage rates, relatively high assist rates and limited defensive efficiency.	’13-14 season (both Denver Nuggets and Washington Wizards) of Andre Miller ’14-15, ’15-16 (for both Miami Heat and Memphis Grizzlies), and ’17-18 seasons of Mario Chalmers
Inefficient scorers	Key characteristic of this cluster are their relatively high rebounding rates, usages rates and significantly poor offensive efficiencies.	’14-15 and ’15-16 seasons of Josh Smith ’17-18 season of Dejounte Murray

3p capable bigs	These players have modest usage and high rebound rates and relatively high assist rates. Their offensive efficiency is higher than average with more than half of their shots coming from the restricted area. Despite their offensive efficiency and high 3-point shot accuracy, they utilise the 3-point shot less than many other clusters.	’14-15, ’16-17, and ’17-18 seasons of Ersan Ilyasova ’17-18 season (both Chicago Bulls and New Orleans Pelicans) of Nikola Mirotic
D&D (defense and distribute)	These players boast very high assist and steal rates along with relatively high usage rates, yet they utilise 3-points infrequently and have low rebound rates and overall efficiency.	’14-15 season (both Washington Wizards and Sacramento Kings) of Andre Miller ’17-18 season (both Cleveland Cavaliers and Miami Heat) of Dwayne Wade
Snipers	These players utilise the 3-point shot significantly and with deadly efficiency. On average they also have mediocre assist, rebound, and steal rate but their defining quality is their 3-point shot.	’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Klay Thompson ’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of J.J. Redick
2nd tier bigs	These players have an elite combination of block, steal, and rebound rates. Their offensive efficiency is respectable but not stunning and their usage rate is mediocre. On average, their shot selection is heavily biased towards painted area and midrange.	’13-14, ’14-15, and ’15-16 seasons of Kevin Garnett ’14-15, and ’15-16 seasons of Roy Hibbert
Scorer bigs	This cluster is defined by their mediocre usage rate, high preference to shoot from the painted area, high offensive efficiency and modest rebound rates.	’13-14, ’14-15 (both New York Knicks and Dallas Mavericks), and ’15-16 seasons of Amar’e Stoudemire ’13-14, ’14-15, ’15-16, and ’16-17 seasons of David Lee
Brick layers	Do not worry if you have never heard of the “notable members” of this group. With their high usage rate and incredibly inefficient offensive ratings, they are the stuff of nightmares for hardcore fans.	’14-15 season of Gary Harris ’13-14 season of Earl Watson
Pass first	This cluster is characterised by their high assist rates and low usage rates meaning they shoot less than average yet pass a lot more. Their primary role is setting up the offensive flow. They also have reasonably high steal rates.	’15-16 season of Jeff Teague ’15-16 season (both Minnesota Timberwolves and San Antonio Spurs) of Andre Miller
Clamps / shackles	These players are the defensive specialists. Their key attribute is the high steal and block rates. Their offensive usage, rebound, and assist rates are all lower than average, indicating a specialist role for defense.	’13-14 Metta World Peace ’13-14, ’14-15, ’15-16, ’16-17, and ’17-18 seasons of Tony Allen
Vanguards	This cluster is characterised by their incredibly low offensive usage rate, very high rebound and block rates, efficient yet low volume offense, and heavy bias towards operating from within the restricted area.	’13-14, ’14-15, ’16-17, and ’17-18 seasons of Ian Mahimi ’13-14, ’14-15, ’16-17, and ’17-18 seasons of Omer Asik

We think it is worth discussing our clustering outputs by applying a temporal lens as well. Some players have been assigned to different clusters in different seasons or even during the same season in different teams (this can only happen if a player is traded to another team or bought out / released by his team and then signed by another team in mid-season). We had also briefly alluded to this feature of our solution design in Chapter 3. One of the key points in our design is its ability to capture the evolution of a player and/or assess how he fits in a new environment.

Some notable examples where our design has captured key changes in playing styles or efficiencies were:

- Andre Drummond, in ’13-14 and ’14-15 seasons is classified as a stereotypical ‘Bruiser’. As he expands his offensive game outside of the restricted area in ’15-16 and ’16-17 seasons, his offensive usage rate and assist rates increase whilst his rebounding rate and scoring efficiency largely remain the same. But the expanded offensive arsenal and subtle drops in his block rates are enough to elevate his ’15-16 and ’16-17 seasons to the ‘Elite big’ category. In the fifth year of our sample (’17-18), we see him regressing to the ‘Bruiser’ category. With the addition of Blake Griffin to the team, Drummond’s real estate is once again confined to the restricted area as evidenced by his shot selection and increase in block rates. Therefore, the clustering algorithm reassigns him to ‘Bruiser’ cluster.
- Blake Griffin has been a divisive figure in NBA circles since his rookie season. Apparently, our clustering solution is indecisive about him as well. After oscillating between ‘Elite bigs’ and ‘Jack of all trades’ clusters during ’13-14, ’14-15, ‘15-16, and ’16-17 seasons in LA Clippers, he gets traded to Detroit Pistons in midseason during ’17-18. His role, as many NBA fans still remember, has shifted drastically and he handled a lot of playmaking duties in his first year, boasting career his high in assists. We capture this change and classify him under ‘Offensive juggernauts’ cluster for the 25 games he played for the Pistons in ’17-18 season.
- Carmelo Anthony, another divisive figure, is classified as a ‘Swiss army knife’ alongside the likes of LeBron James and Kevin Durant in his ’13-14 campaign. He then gets moved to ‘Offensive Juggernauts’ due to his

decreased contribution in everything but offensive stats and remains in that cluster from ’14-15 to ’16-17 season. He then gets reclassified as a ‘Mid-tier scorer’ reflecting his decreased scoring efficiency and limited contribution in other areas of the game.

- Kawhi Leonard’s ’13-14 and 14’-15 campaign is classified under the ‘Jack of all trades’ cluster reflecting his multi-faceted offensive and defensive skills. He then elevates his game and for the remaining 3 years in our dataset he is labelled as a ‘Swiss Army Knife’ player.
- Giannis Antetokounmpo is labelled as a ‘Clamp/shackle’ in his rookie season in ’13-14. He then gets relabelled as a ‘Jack of all’ in the next two years, before elevating into ‘Elite big’ cluster in ’16-17 and ’17-18 campaigns. Due to his increased assist rates in following seasons (’18-19 and forward is not included in our research scope), he would most likely be a Swiss Army Knife.

5.2 Game outcome predictions

Following our clustering analysis, we have, across the five seasons in our dataset, approximate team configurations (based on the clusters identified) for each of the 6,150 games in these seasons. This input set is in accordance with our key design principles stated in Chapter 1[8]. Due to time and resource constraints, we were able to train models using only a small subset of the clustering outputs we produce, details of this selection process have been discussed in detail in the Chapter 5.1 -*Clustering Analysis.* The highest prediction accuracy across all eight input sets has been

[8] These design principles were defined as follows:
i. The five traditional positions do not adequately capture player tendencies and therefore a new way of classifying players is needed; and
ii. Players occupying same traditional positions can have significantly different playing styles and therefore the correct mix of components need to be found for optimal results (e.g., guards who tend to drive frequently synergise better with centres and power forwards who are capable three-point shooters rather than low-post scorers).

achieved by the ANNs optimised by genetic algorithms (average prediction accuracy of 75.64%), followed by SVMs (average accuracy 71.43%). Table – 4 presents a breakdown of prediction accuracy for each prediction method per each of our 8 input sets discussed in detail.

Table 4. Breakdown of Prediction Accuracy By Predictor And Input

Model ID	**Dataset**	**Clustering Algorithm**	**Distance metric**	***k* value**	**GA ANN accuracy** (%)	**SVM accuracy** (%)	**Naïve predictor (home team win %)**	**Human experts (%)**
1	Full dataset with PCA	c-means	Cosine	10	76.52	72.76	N/A	N/A
2	Subset 2 with PCA	k-means	Euclidean	26	76.29	71.00		
3	Subset 1 with PCA	Affinity propagation	Squared Euclidean	101	76.07	70.61		
4	Subset 1 with PCA	c-means	Cosine	10	75.81	72.37		
5	Full dataset with PCA	k-means	Euclidean	25	75.64	72.3		
6	Subset 3 with PCA	k-means	Euclidean	23	75.35	73.92		
7	Subset 1 with PCA	k-means	Euclidean	27	75.39	72.5		
8	Subset 1 with PCA	Affinity propagation	Cosine	80	73.05	65.99		
		Average accuracy			75.64	71.43	58	~ 65-68

As seen in Table -4, for *k*-means and *c*-means outputs, highest prediction accuracy region overlaps with the high values of clustering validity indices, reinforcing our initial prediction that these indices are a good proxy of our models' capability to explain the underlying groups. An outcome that we did not expect was the ability of Subset 1 to enable the training of such high-accuracy prediction models. As discussed before, there is a significant amount of research on finding new player statistics to supplement the traditional box score metrics. Despite the widely recognised shortcomings of traditional box score metrics, they are still able to capture the differences in playing styles to a reasonable extent; after all, there is a reason why they have withstood the test of time.

The relatively small range of prediction accuracies, especially for the genetic algorithm optimised ANNs merits further discussion. In addition to the player

clustering outcomes, we also added three team level attributes to our input space: rest days for teams, dummy variables for calendar months, and the winning percentage of both teams to date. We are conscious that the winning percentage may and actually has been used as a standalone predictor in many other studies. Including this variable without any mitigation, therefore, runs the risk of contaminating our prediction results. To account for this, we double check our models by:

- Training all of the 8 models in Table – 4 without this attribute; and
- Training models (i.e., simple predictors) using GA ANN and SVM with only team-level inputs (i.e., team win percentages, rest days, and calendar month variables).

We find that the accuracy of our models drops approximately 8% when we exclude the winning percentage as an attribute. This is to be expected and we believe it to be quite acceptable given the fact that models trained using only winning percentage and rest days as inputs via GA ANN achieve prediction accuracy of 72%. We further explain this robustness check and other robustness checks employed in Chapter 5.3 – Robustness Checks.

SVMs perform admirably well in terms of prediction accuracy if we take into account the cost of training. Across all datasets, their prediction accuracy is, on average, only 4.21% lower than GA ANNs. Since training an SVM takes approximately two orders of magnitude less computing time[9] than evolving 500 individuals over 300 generations for our GA ANN, their performance is remarkable. As mentioned previously, we also experiment with randomly searching through a pre-defined hyperparameter space for ANNs is the third prediction method we apply and as seen in Table – 4, it compares poorly against the other two methods. Yet, it

[9] The computing time comparisons are performed based on the CPU consumption of the workstation we used to train both GA ANN and SVMs.

requires an order of magnitude more computing time than SVMs, marking it as an inefficient and ineffective option among the three.

It should be noted that we only focus on the accuracy of our predictors and so far and have refrained from discussing their performance in regards to precision or recall metrics. Whilst the performance ranking of the top eight models do not change when we rank them by precision or recall, we still think it is worth to analyse these metrics as well. This is because whilst one may think the cost of a false positive or a false negative (i.e., predicting home team win when away team wins or predicting away team win when the home team wins) is not necessarily significantly different, one would be gravely wrong. In NBA games, the distribution of home wins is not uniform across teams. Some teams win a disproportionate percentage of both home and away games whilst majority of the teams win significantly less frequently on the road. Table – 5 below presents the breakdown of home and away win rates of each team across the 5 seasons in our dataset.

Table 5. Home And Away Win Percentages of Teams Across Our Dataset

Team	Percentage of games won at home	Percentage of games won away	Difference	Team	Percentage of games won at home	Percentage of games won away	Difference
GSW	83%	71%	12%	**UTA**	58%	41%	17%
SAS	82%	60%	22%	**CHO**	57%	39%	19%
HOU	73%	59%	14%	**NOP**	57%	37%	20%
OKC	73%	51%	22%	**DAL**	55%	42%	13%
TOR	72%	57%	15%	**DEN**	55%	36%	19%
LAC	70%	56%	14%	**DET**	54%	35%	19%
POR	70%	46%	24%	**MIL**	51%	35%	16%
CLE	70%	49%	20%	**MIN**	47%	32%	15%
IND	68%	43%	25%	**BRK**	43%	30%	13%
MIA	63%	46%	17%	**ORL**	43%	24%	19%
WAS	61%	47%	14%	**PHO**	42%	33%	9%
ATL	61%	43%	18%	**NYK**	41%	30%	12%
MEM	60%	43%	18%	**SAC**	41%	32%	9%
BOS	60%	48%	11%	**PHI**	37%	25%	12%
CHI	60%	42%	18%	**LAL**	37%	25%	12%

It is obvious that a false negative for a GSW home game (which happens only in 17% of GSW home games in our dataset) would be a much worse prediction than a false negative for a MIL home game (which happens in 49% of MIL home games in our dataset). Vice versa, correctly predicting a GSW win at home is much easier than correctly predicting an ORL away win (which only happens in 24% of times). And then there are games (e.g., CLE @ MIL, where CLE wins 49% of its away games and MIL wins 51% of its home games) that where misclassification is almost as probable as classification, to put it simply. For this reason, the cost of a false positive or a false negative is largely dependent on the teams playing, as any betting person can easily tell! To assess these metrics at a team level, we use the top three models in terms of accuracy from Table – 4 and analyse their precision, recall, and F1 scores at team levels, in Table – 6 below.

Table 6. Team Level Analysis of Precision, Recall, And F1 Scores Of Top 3 (In Terms Of Accuracy) Models

Teams	Model 1			Model 2			Model 3		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ATL	0.72	0.87	0.79	0.77	0.81	0.79	0.74	0.84	0.79
BOS	0.73	0.89	0.80	0.75	0.83	0.79	0.73	0.87	0.79
BRK	0.75	0.56	0.64	0.73	0.53	0.61	0.67	0.67	0.67
CHI	0.66	0.77	0.71	0.70	0.74	0.72	0.68	0.83	0.75
CHO	0.71	0.79	0.75	0.73	0.78	0.76	0.70	0.88	0.78
CLE	0.79	0.92	0.85	0.79	0.88	0.83	0.80	0.92	0.86
DAL	0.68	0.74	0.71	0.74	0.67	0.70	0.69	0.77	0.73
DEN	0.70	0.70	0.70	0.71	0.64	0.67	0.68	0.77	0.72
DET	0.71	0.84	0.77	0.78	0.72	0.75	0.68	0.84	0.75
GSW	0.86	0.99	0.92	0.87	0.98	0.92	0.86	0.98	0.92
HOU	0.79	0.97	0.87	0.81	0.93	0.86	0.78	0.97	0.87
IND	0.79	0.91	0.85	0.80	0.82	0.81	0.79	0.89	0.84
LAC	0.78	0.96	0.86	0.83	0.92	0.87	0.80	0.94	0.87
LAL	0.65	0.49	0.56	0.62	0.37	0.47	0.63	0.53	0.58
MEM	0.72	0.89	0.80	0.74	0.82	0.78	0.71	0.85	0.78
MIA	0.74	0.90	0.81	0.76	0.81	0.78	0.78	0.86	0.82
MIL	0.70	0.76	0.73	0.72	0.73	0.73	0.69	0.78	0.73
MIN	0.74	0.72	0.73	0.73	0.72	0.73	0.70	0.76	0.73

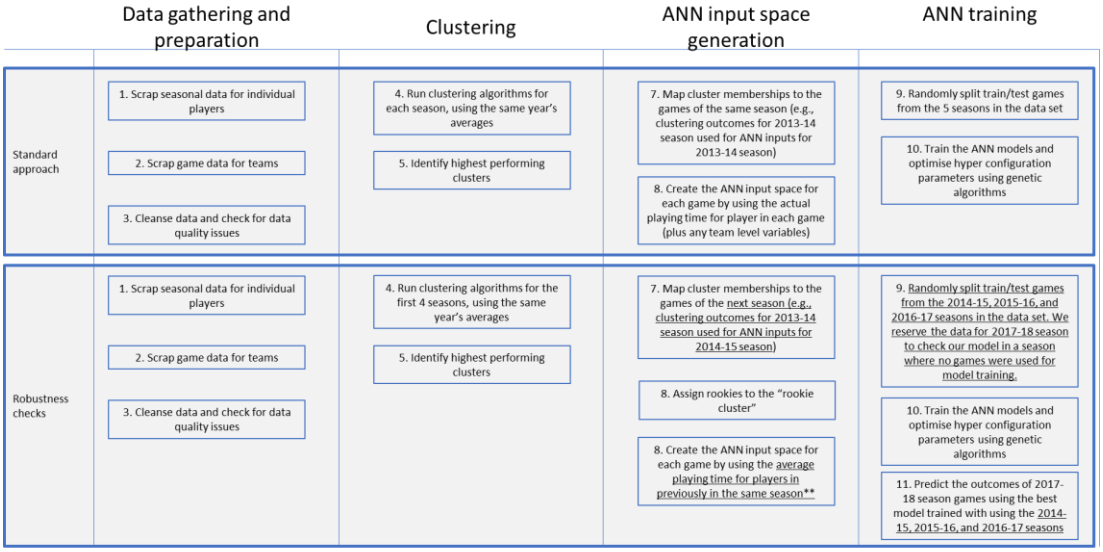
NOP	0.72	0.63	0.67	0.72	0.64	0.68	0.66	0.79	0.72
NYK	0.70	0.75	0.73	0.72	0.62	0.67	0.71	0.61	0.66
OKC	0.79	0.88	0.83	0.81	0.84	0.83	0.80	0.92	0.86
ORL	0.63	0.60	0.62	0.61	0.56	0.58	0.59	0.72	0.65
PHI	0.72	0.61	0.66	0.70	0.41	0.52	0.71	0.72	0.71
PHO	0.62	0.70	0.66	0.68	0.66	0.67	0.63	0.71	0.67
POR	0.80	0.93	0.86	0.80	0.92	0.86	0.79	0.94	0.86
SAC	0.67	0.62	0.64	0.63	0.48	0.54	0.68	0.51	0.59
SAS	0.85	0.98	0.91	0.86	0.95	0.90	0.85	0.98	0.91
TOR	0.76	0.98	0.86	0.80	0.94	0.86	0.76	0.98	0.86
UTA	0.75	0.78	0.77	0.81	0.70	0.75	0.72	0.82	0.77
WAS	0.69	0.87	0.77	0.70	0.79	0.75	0.68	0.80	0.74

It is evident that accuracy, precision, recall, and F1 metrics differ significantly per team. Our predictors, whilst on average outperforming human experts and majority of the existing predictors in previous studies, are still significantly vulnerable to the unpredictability of some teams.

5.3 Robustness checks

We recognise that our approach has certain limitations, some of which have been mentioned above. We perform some experiments to address these limitations and review whether our approach still achieves our goal. We provide an overview of our end-to-end process from data gathering to ANN model training and how that end-to-

end process changes in our robustness checks in Figure -7 below.



* We considered using pre-season games to cluster rookies but a brief analysis showed significant variation between rookie performance between actual NBA games and preseason games.
** We calculate the average playing time three times in each season – after the first 10, 30, and 50 games. This allows us to quickly adjust / correct assumed minute allocations after players change teams and for rookies.

Figure 7. Robustness Checks Applied Across Multiple Parts of The Process

Also, in Table -7 below, we list the limitations and how our robustness checks (where available) address these limitations.

Table 7. Limitations, Robustness Checks Applied, and Their Results

Limitation	**Rationale and Robustness check applied**	**Results and implications**
1. ANN inputs include data from the future (e.g., player stereotypes for the 2014-15 season are calculated with this season's averages. Output of this clustering is used to predict the same season's results)	We employ a staggered approach between clustering and ANN. To predict a game from *season x* we use *x-1 season's* clustering data. We use 2013-14 (first season in our data set) only to calculate cluster memberships. These are used to predict 2014-15 seasons' games. Clustering outcomes of 2014-15 is used for predictions of the 2015-16 season and the clustering outputs of 2015-16 to predict the 2016-17 season. We do not run clustering on the 2016-17 season and leave the 2017-18 games out of test/train data set. We use the 2017-18 season to independently assess the prediction powers of our model.	Our models trained with this approach achieve only slightly lower prediction accuracy scores (overall prediction accuracy of 75.03% compared to 76.3% of our initial approach). We think that the prediction accuracy of the robustness checks may be artificially inflated because of the smaller data set. This is because we expected a slightly larger drop in the prediction accuracy due to the information loss.
2. Our test/train split draws games randomly from all 5 seasons in our data set. We do not check the predictive		When we run our new model for the 2017-18 season (which was not used in its

capability of our models with a season that was not used for model training.		training at all), we achieve a 70.65% prediction accuracy across that season. We think that these robustness check support our approach to use player stereotypes for prediction.
3. Our ANN input space is calculated by using the actual minutes played by each player.	We retrain our models, using the average playing time for players in previously in the same season. We calculate the average playing time three times in each season – after the first 10, 30, and 50 games. This allows us to quickly adjust / correct assumed minute allocations after players change teams and for rookies.	We see no significant changes in the prediction accuracy of our models. We admit that the 10, 30, and 50 game thresholds to recalculate the minute allocations are chosen arbitrarily and the frequency can be adjusted for increased precision. We also appreciate that in-game events such as injuries or ejections can lead to significant discrepancies between the model inputs and actual minutes played. However, these can only be captured by in-game (e.g., possession by possession) prediction engines and this is beyond the scope of our research.
4. Our ANN input space uses team-level "Win %" statistics in addition to player stereotypes	We retrain our models without the "Win %" variables. These are proxies for the teams' power rankings, encapsulating all their wins and losses in a given season up until the game date. We remove this variable for both home and away teams and retrain our models.	As we expected, prediction accuracy of our models drops to: - 67.7% for the *k-means with 26 clusters (from 76.29%);* - 66.1% for the *k-means with 25 clusters (from 75.36%); and* - *66.3 for the c-means with 10 clusters (from 76.52%).* This shows that whilst the "Win %" contains significant information, our models can still compete with human experts without this variable, validating our approach.
5. For comparison, we apply the robustness check #2 to the simple predictor as defined in Chapter 5.2	We apply the robustness checks #2 to a model trained using only team level statistics (i.e., win percentages, rest days, and calendar month of the game variables). We reserve the last season's (i.e., 2017-18) data for assessing the prediction	The models produced by team level variables achieve *67.34%* accuracy for the 2017-18 season. This is on par with our best performing model trained without these team level variables.

	performance. We run predictions across the entire 2017-18 season and compare the prediction performance against our approach so that our models trained with the robustness checks are benchmarked against the simple predictor with the same robustness checks applied (i.e., not the 72.02% prediction accuracy achieved with the simple predictor without the robustness checks).	
6. Our approach cannot capture any new skills or playing styles developed over the summer and training camps.	N/A	N/A
7. Our approach is built on the assumption that the underlying positive and/or negative synergies between clusters will not significantly change over time.	N/A	This limitation is, to a certain extent, mitigated by using isolating the 2017-18 season from the test/training data set.
8. Our prediction algorithm does not discriminate between "safe to predict" vs "difficult to predict" games.	N/A	N/A
9. Our clustering methods do not take into account the "contract year" variable.	Stiroh (2007) analyses the impact of contract years (i.e., the last year of a player's existing contract) on player performance, to see whether players have greater incentive and drive to showcase their performance with the hope of securing a better contract for future. He finds a non-trivial difference in some players' contract year performance,	We opt to exclude this as we cannot predict the nature of the performance change (e.g., a player moving to a more team-friendly playing style or just trying to do more of what they already do).
10. Our predictors do not consider the coaches'	The impact of a coach on a team's performance is difficult to measure and also is an ongoing debate. (Tarlow, 2012), for example shows that a coach's	Due to the lack of any research, to the best of our knowledge, establishing proxy measurements on coaches influence on

experience and / or success.	post-season experience is not an indicator of team success.	game outcomes, we opted to omit this variable.
11. Our predictors do not take into account the travel teams make between consecutive games.	Travel is a key feature of life in the NBA. Players sometimes travel across 3 time zones to play back-to-back games in consecutive days. Over the course of a 82 game season from October to April, travel will surely take its toll, even on the players with mental and physical stamina levels comparable to Atlas. We use the number of rest days since the last game in our feature set for predictions but a day of rest at home is surely not the same as a day of rest during travel.	However, gathering and parsing the data on the time/distance travelled was a non-trivial task and we assessed the costs would outweigh the benefits.

CHAPTER 6

CONCLUSION

In this thesis, we aimed to find a reliable method to predict the outcome of NBA games. There is already a significant and bountiful literature related to this problem, mainly because sport events are inherently unpredictable due to the large number of components and variables related as well as the non-linear interactions among them. This thesis helps build the theory on a significant gap in the literature by approaching the prediction problem from a complex system approach. As we discuss in Chapter 2, the majority of the existing research on game outcome prediction uses players' individual statistics aggregated at team level. This, we believe has severe limitations: the individual statistics are the outcomes of all the complex relationship happening throughout the game, but they may not necessarily yield information on the relationships themselves. We agree with Lames & McGarry's (2007) view that these are *ex post facto* variables and not necessarily reliable measures of team performance either. Following a slightly similar approach to Lutz (2012), Oh et. al (2015), and Kuehn (2017), we identify different player stereotypes, completely agnostic of how players are labelled according to the five traditional positions in basketball.

By identifying the different playing styles across the league, we aimed to capture how well certain team configurations worked by generating positive synergies, leading to more wins. We use numerous clustering techniques to identify which clustering configuration would generate the best input space for predictors.

We adopt this heuristic approach because there are no established methods in the existing literature on the player stereotypes in the NBA, or even in basketball in

general. We opt to use k-means, c-means, and Affinity Propagation together because c-means is a nice complement to k-means with its ability to capture players exhibiting high-levels of phenotype plasticity, and Affinity Propagation complements both of these algorithms with its ability to infer the number of underlying clusters.

We must stress though that we are conscious these are probably neither the best algorithms nor the clustering configurations to capture the player stereotypes. However, the prediction results show that our principles and approach is sound, as we manage to outperform (in terms of prediction accuracy) human experts and majority of the existing research in similar settings: our best model achieves a ~76% prediction accuracy across the span of five NBA seasons. This extensive sample size is another key feature of our research; by analysing different roster configurations over the course of five seasons, we are confident that we have limited the cyclical impact of data in our dataset.

We employ genetic algorithms to optimise the hyperparameter configurations of ANNs, a common use case in this setting. However, another potential use case for genetic algorithms are feature selection, as shown by (Vafaie & De Jong, 1992). Implementing another layer of genetic algorithms could, in theory, also help mitigate the input sparsity matrix issue discussed earlier, by limiting the dimensionality of our input space. Yet for the same reasons we discussed above, we think the cost of implementation and increasing the complexity of our solution was not commensurate with the benefits we would get and therefore opted not to adopt this use case. Whilst (Whitley et al., 1990) have shown that employing adaptive mutation rates enable better population diversity, due to having a relatively limited pool of potential genes and the measures we implement to avoid clones in the population, we think constant mutation rates are a reasonable choice. We also forego k-fold cross

validation for our GA ANN implementation to limit the already significant computing time required for the GA ANNs.

In the light of all the limitations discussed above, the accuracy, precision, and recall of our high-performing predictors are promising. We think our research strongly reinforces the need to think the intrateam interactions and the synergies (both positive and negative) to predict team performance. We show that by analysing player propensities and tendencies through prototype heuristics, it is possible to extract rich information on the player stereotypes. Using these classes, one can predict the outcomes of basketball games and gain better insight on how to build more successful teams. A side benefit of our research is our findings on the prediction performance of a very limited set of team-level attributes. By only using the winning percentages of the teams in a particular season, and the number of rest days each team had before the game, we were able to train simple predictors with c. 72% prediction accuracy, outperforming many existing research and the human expert benchmarks. This unexpected result shows that, win rates, the ultimate aggregation of individual statistics to team-level, performs at least as well as the team-level aggregated statistics.

APPENDIX A: PYTHON CODE FOR CLUSTERING

```
#!/usr/bin/env python
# coding: utf-8
#read data from csv, define sample size and full feature set for PCA
# import libraries
from __future__ import division, print_function
import pandas as pd
from sklearn.decomposition import PCA
from scipy.spatial import distance
import sklearn
from sklearn.cluster import AffinityPropagation
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
from sklearn.decomposition import PCA
import skfuzzy as fuzz
import qgrid
import time
get_ipython().run_line_magic('config', 'IPCompleter.greedy=True')
```

```
print('Library import complete')
#generate source dataframe
df=pd.read_csv('20190224v1.csv', sep=',',header=0)
   #print(testarray)
df['MPG']=df['MP']/df['G']
df['GS%']=df['GS']/df['G']
df.loc[1:3]
sample=2552
print('Initial dataframe generated')
#check if you are using all data points
len(df)==sample
#define features for full set and subsets
#previous version of features - used to include age and salary
#features=['Age', 'MPG', 'PER','TS%', '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%',
'AST%', 'STL%', 'BLK%','TOV%', 'USG%', 'GS%', 'FG', 'FGA', 'FG%', '3P', '3PA',
'3P%','2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%', 'ORB', 'DRB', 'TRB','AST',
'STL', 'BLK', 'TOV', 'PF', 'PS/G', 'Above the Break 3-Usage','Mid-Range-Usage', 'In
The Paint (Non-RA)-Usage','Restricted Area-Usage', 'Right Corner 3-Usage', 'Left
Corner 3-Usage','Backcourt-Usage', 'Above the Break 3 %', 'Mid-Range %','In The
Paint (Non-RA) %', 'Restricted Area %', 'Right Corner 3 %','Left Corner 3 %',
'Backcourt %', 'SalaryLog']
#ss1=['PSTK','Age','MPG','GS%','FG','FGA','FG%','3P','3PA','3P%','2P','2PA','2P%','
FT','FTA','FT%','ORB','DRB','TRB','AST','STL','BLK','TOV','PS/G','SalaryLog']
#ss2=['PSTK','Age','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL
%','BLK%','TOV%','USG%','GS%','MPG','eFG%','SalaryLog']
```

#ss3=['PSTK','MPG','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','TOV%','USG%','Above the Break 3-Usage','Mid-Range-Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range %','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3 %','Backcourt %','SalaryLog']

#fss1=['Age','MPG','GS%','FG','FGA','FG%','3P','3PA','3P%','2P','2PA','2P%','FT','FTA','FT%','ORB','DRB','TRB','AST','STL','BLK','TOV','PS/G','SalaryLog']

#fss2=['Age','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','TOV%','USG%','GS%','MPG','eFG%','SalaryLog']

#fss3=['MPG','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','TOV%','USG%','Above the Break 3-Usage','Mid-Range-Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range %','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3 %','Backcourt %','SalaryLog']

features=['MPG', 'PER','TS%', '3PAr', 'FTr', 'ORB%', 'DRB%', 'TRB%', 'AST%', 'STL%', 'BLK%','TOV%', 'USG%', 'GS%', 'FG', 'FGA', 'FG%', '3P', '3PA', '3P%','2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%', 'ORB', 'DRB', 'TRB','AST', 'STL', 'BLK', 'TOV', 'PF', 'PS/G', 'Above the Break 3-Usage','Mid-Range-Usage', 'In The Paint (Non-RA)-Usage','Restricted Area-Usage', 'Right Corner 3-Usage', 'Left Corner 3-Usage','Backcourt-Usage', 'Above the Break 3 %', 'Mid-Range %','In The Paint (Non-RA) %', 'Restricted Area %', 'Right Corner 3 %','Left Corner 3 %', 'Backcourt %']

```
ss1=['PSTK','MPG','GS%','FG','FGA','FG%','3P','3PA','3P%','2P','2PA','2P%','FT','FT
A','FT%','ORB','DRB','TRB','AST','STL','BLK','TOV','PS/G']
ss2=['PSTK','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK
%','TOV%','USG%','GS%','MPG','eFG%']
ss3=['PSTK','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK
%','TOV%','USG%','GS%','MPG','eFG%','Above the Break 3-Usage','Mid-Range-
Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-
Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range
%','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3
%','Backcourt %']
fss1=['MPG','GS%','FG','FGA','FG%','3P','3PA','3P%','2P','2PA','2P%','FT','FTA','FT
%','ORB','DRB','TRB','AST','STL','BLK','TOV','PS/G']
fss2=['PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','T
OV%','USG%','GS%','MPG','eFG%']
fss3=['PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','T
OV%','USG%','GS%','MPG','eFG%','Above the Break 3-Usage','Mid-Range-
Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-
Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range
%','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3
%','Backcourt %']
print('Subset feature generation complete')
# Subset dataframe generation
dfss1 = pd.DataFrame(data = df, columns = ss1)
dfss2 = pd.DataFrame(data = df, columns = ss2)
dfss3 = pd.DataFrame(data = df, columns = ss3)
```

```
print(dfss1.loc[1:3])
print('Subset1 dataframe generation complete')
print(dfss2.loc[1:3])
print('Subset2 dataframe generation complete')
print(dfss3.loc[1:3])
print('Subset3 dataframe generation complete')
print('Subset dataframe generation complete')
# Generate dataframe to store clustering output summaries
dfo=pd.DataFrame(columns = ['InputArray','Algo','n_used','Distance
Function','Silhouette','Calinski-Harabaz Index','Davies Bouldin Score','FPC','TBC'])
#define C-means output arrays
output=[]
cmmembership=[]
fss3=['PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK%','T
OV%','USG%','GS%','MPG','eFG%','Above the Break 3-Usage','Mid-Range-
Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-
Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range
%','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3
%','Backcourt %']
ss3=['PSTK','PER','TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','STL%','BLK
%','TOV%','USG%','GS%','MPG','eFG%','Above the Break 3-Usage','Mid-Range-
Usage','In The Paint (Non-RA)-Usage','Restricted Area-Usage','Right Corner 3-
Usage','Left Corner 3-Usage','Backcourt-Usage','Above the Break 3 %','Mid-Range
%','In The Paint (Non-RA) %','Restricted Area %','Right Corner 3 %','Left Corner 3
%','Backcourt %']
```

```
print(len(fss1))
print(len(fss2))
print(len(fss3))
print(len(features))
arraynax = arraynax.astype(np.float)
# # Standardise and PCA full dataset
#1 check  any null values and replace them with 0
df.isnull().values.any()
dfna=df.fillna(0)
dfna.isnull().values.any()
dfnaf = pd.DataFrame(data = dfna, columns = features)
#2 separate Features and item ID
arraynax=dfna.loc[:sample,features].values
arraynay=dfna.loc[:sample,'PSTK'].values
   #print(arraynax)
   #print(arraynay)
#3 standardise features
dfnax = StandardScaler().fit_transform(arraynax)
   #print(arraynax)
#4 reshape arraynay for dataframe generation
   #print(arraynax.shape)
   #print(arraynay.shape)
arraynay.reshape(2552,1)
   #print(arraynay.shape)
```

```
#5 generate the normalised dataframe
dfnanor = pd.DataFrame(dfnax,columns=features)
dfnanor['PSTK'] = arraynay
   #print(dfnanor)
#6 generate 30 PCs
pca1 = PCA(n_components=15)
principalComponents1 = pca1.fit_transform(dfnax)
dfnaPCA = pd.DataFrame(data = principalComponents1, columns = ['PC 1', 'PC 2',
'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7', 'PC 8', 'PC 9', 'PC 10', 'PC 11', 'PC 12', 'PC 13',
'PC 14', 'PC 15'])#, 'PC 16', 'PC 17', 'PC 18', 'PC 19', 'PC 20','PC 21', 'PC 22', 'PC 23',
'PC 24', 'PC 25', 'PC 26', 'PC 27', 'PC 28', 'PC 29', 'PC 30'])
   #print(dfnaPCA)
#7 generate the dataframe for PCA
   #print(dfnaPCA)
   #interim1=dfna.loc[0:sample,'PSTK']
   #print(interim1)
dfnaPCA['PSTK'] = arraynay
   #print(dfnaPCA)
#8 show explained variance for each PC
pca1.explained_variance_ratio_
plt.plot(100*np.cumsum(pca1.explained_variance_ratio_))
plt.xlabel('number of components')
plt.xticks(np.arange(0, 18, 1.0))
plt.ylabel('cumulative explained variance');
```

```
#9 Correlation heatmap to see hotspots in variables
corr = dfnanor.corr()
plt.figure(figsize=(16, 16))
sns.heatmap(corr, cmap="PiYG", xticklabels=corr.columns.values,
yticklabels=corr.columns.values)
# # Standardise and PCA SS1
#1 check  any null values and replace them with 0
dfss1.isnull().values.any()
dfss1na=df.fillna(0)
dfss1na.isnull().values.any()
dfss1f = pd.DataFrame(data = dfss1na, columns = fss1)
#2 separate Features and item ID
arrayss1nax=dfna.loc[:sample,fss1].values
arrayss1nay=dfna.loc[:sample,'PSTK'].values
   #print(arraynax)
   #print(arraynay)
#3 standardise features
dfss1nax = StandardScaler().fit_transform(arrayss1nax)
   #print(arraynax)
#4 reshape arraynay for dataframe generation
   #print(arraynax.shape)
   #print(arraynay.shape)
arrayss1nay.reshape(2552,1)
   #print(arraynay.shape)
```

```
#5 generate the normalised dataframe
dfss1nanor = pd.DataFrame(dfss1nax,columns=fss1)
dfss1nanor['PSTK'] = arrayss1nay
   #print(dfnanor)

#6 generate 20 PCs
pcass1 = PCA(n_components=7)
principalComponentsss1 = pcass1.fit_transform(dfss1nax)
dfss1naPCA = pd.DataFrame(data = principalComponentsss1, columns = ['PC 1', 'PC
2', 'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7'])#, 'PC 8', 'PC 9', 'PC 10', 'PC 11', 'PC 12', 'PC
13', 'PC 14', 'PC 15', 'PC 16', 'PC 17', 'PC 18', 'PC 19', 'PC 20'])
   #print(dfnaPCA)

#7 generate the dataframe for PCA
   #print(dfnaPCA)
   #interim1=dfna.loc[0:sample,'PSTK']
   #print(interim1)
dfss1naPCA['PSTK'] = arrayss1nay
   #print(dfnaPCA)

#8 show explained variance for each PC
pcass1.explained_variance_ratio_
plt.plot(100*np.cumsum(pcass1.explained_variance_ratio_))
plt.xlabel('number of components')
plt.xticks(np.arange(0, 20, 1.0))
```

```
plt.ylabel('cumulative explained variance');


#9 Correlation heatmap to see hotspots in variables
corr = dfss1nanor.corr()
plt.figure(figsize=(16, 16))
sns.heatmap(corr, cmap="PiYG", xticklabels=corr.columns.values,
yticklabels=corr.columns.values)
# # Standardise and PCA SS2
#1 check  any null values and replace them with 0
dfss2.isnull().values.any()
dfss2na=df.fillna(0)
dfss2na.isnull().values.any()
dfss2f = pd.DataFrame(data = dfss2na, columns = fss2)


#2 separate Features and item ID
arrayss2nax=dfna.loc[:sample,fss2].values
arrayss2nay=dfna.loc[:sample,'PSTK'].values
   #print(arraynax)
   #print(arraynay)


#3 standardise features
dfss2nax = StandardScaler().fit_transform(arrayss2nax)
   #print(arraynax)


#4 reshape arraynay for dataframe generation
```

```
    #print(arraynax.shape)
    #print(arraynay.shape)
arrayss2nay.reshape(2552,1)
    #print(arraynay.shape)

#5 generate the normalised dataframe
dfss2nanor = pd.DataFrame(dfss2nax,columns=fss2)
dfss2nanor['PSTK'] = arrayss2nay
    #print(dfnanor)

#6 generate 15 PCs
pcass2 = PCA(n_components=8)
principalComponentsss2 = pcass2.fit_transform(dfss2nax)
dfss2naPCA = pd.DataFrame(data = principalComponentsss2, columns = ['PC 1', 'PC
2', 'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7', 'PC 8'])#, 'PC 9', 'PC 10', 'PC 11', 'PC 12', 'PC
13', 'PC 14', 'PC 15'])
    #print(dfnaPCA)

#7 generate the dataframe for PCA
    #print(dfnaPCA)
    #interim1=dfna.loc[0:sample,'PSTK']
    #print(interim1)
dfss2naPCA['PSTK'] = arrayss2nay
    #print(dfnaPCA)
```

```
#8 show explained variance for each PC
pcass2.explained_variance_ratio_
plt.plot(100*np.cumsum(pcass2.explained_variance_ratio_))
plt.xlabel('number of components')
plt.xticks(np.arange(0, 20, 1.0))
plt.ylabel('cumulative explained variance');

#9 Correlation heatmap to see hotspots in variables
corr = dfss2nanor.corr()
plt.figure(figsize=(16, 16))
sns.heatmap(corr, cmap="PiYG", xticklabels=corr.columns.values,
yticklabels=corr.columns.values)
# # Standardise and PCA SS3
#1 check  any null values and replace them with 0
dfss3.isnull().values.any()
dfss3na=df.fillna(0)
dfss3na.isnull().values.any()
dfss3f = pd.DataFrame(data = dfss3na, columns = fss3)

#2 separate Features and item ID
arrayss3nax=dfna.loc[:sample,fss3].values
arrayss3nay=dfna.loc[:sample,'PSTK'].values
   #print(arraynax)
   #print(arraynay)
```

```
#3 standardise features
dfss3nax = StandardScaler().fit_transform(arrayss3nax)
   #print(arraynax)

#4 reshape arraynay for dataframe generation
   #print(arraynax.shape)
   #print(arraynay.shape)
arrayss3nay.reshape(2552,1)
   #print(arraynay.shape)

#5 generate the normalised dataframe
dfss3nanor = pd.DataFrame(dfss3nax,columns=fss3)
dfss3nanor['PSTK'] = arrayss3nay
   #print(dfnanor)

#6 generate 20 PCs
pcass3 = PCA(n_components=16)
principalComponentsss3 = pcass3.fit_transform(dfss3nax)
dfss3naPCA = pd.DataFrame(data = principalComponentsss3, columns = ['PC 1', 'PC
2', 'PC 3', 'PC 4', 'PC 5', 'PC 6', 'PC 7', 'PC 8', 'PC 9', 'PC 10', 'PC 11', 'PC 12', 'PC 13',
'PC 14', 'PC 15', 'PC 16'])#, 'PC 17', 'PC 18', 'PC 19', 'PC 20', 'PC 21', 'PC 22', 'PC
23', 'PC 24', 'PC 25'])
   #print(dfnaPCA)

#7 generate the dataframe for PCA
```

```
    #print(dfnaPCA)
    #interim1=dfna.loc[0:sample,'PSTK']
    #print(interim1)
dfss3naPCA['PSTK'] = arrayss3nay
    #print(dfnaPCA)

#8 show explained variance for each PC
pcass3.explained_variance_ratio_
plt.plot(100*np.cumsum(pcass3.explained_variance_ratio_))
plt.xlabel('number of components')
plt.xticks(np.arange(0, 25, 1.0))
plt.ylabel('cumulative explained variance');

#9 Correlation heatmap to see hotspots in variables
corr = dfss3nanor.corr()
plt.figure(figsize=(16, 16))
sns.heatmap(corr, cmap="PiYG", xticklabels=corr.columns.values,
yticklabels=corr.columns.values)
# # Generate list of dataframes to be ingested by K-means
listkm=[dfnanor,dfnaPCA,dfss1nanor,dfss1naPCA,dfss2nanor,dfss2naPCA,dfss3nan
or,dfss3naPCA]
listkmstr=['dfnanor','dfnaPCA','dfss1nanor','dfss1naPCA','dfss2nanor','dfss2naPCA','
dfss3nanor','dfss3naPCA']
listkmdic={
    "dfnanor":"Full dataset normalised",
```

```
    'dfnaPCA':"Full dataset PCA'd",
    'dfss1nanor':"Subset 1 - traditional box score dataset - normalised",
    'dfss1naPCA':"Subset 1 - traditional box score dataset - PCA'd",
    'dfss2nanor':"Subset 2 - advanced box score dataset - normalised ",
    'dfss2naPCA':"Subset 2 - advanced box score dataset - PCA'd",
    'dfss3nanor':"Subset 3 - advanced box score dataset and shot placement -
normalised ",
    'dfss3naPCA':"Subset 3 - advanced box score dataset and shot placement - PCA'd
"
}
listnonstd=[dfnaf,dfss1f,dfss2f,dfss3f]
listnonstdstr=['dfnaf','dfss1f','dfss2f','dfss3f']
listnonstddic={
    "dfnaf":"Full dataset",
    'dfss1f':"Subset 1 - traditional box score dataset",
    'dfss2f':"Subset 2 - advanced box score dataset",
    'dfss3f':"Subset 3 - advanced box score dataset and shot placement",
}

# # Define global parameters for K-means code
slicer=2552
maxcluster=35
output=[]
kmeansmemberships = pd.DataFrame([df.PSTK]).transpose()
```

```
# # Define K-means function (runkm) and then run it for all datasets
def pipeline(x):
    for i,k in enumerate(listkm):
        text1=listkmdic.get(listkmstr[i])
        print('Commencing K-means clustering for {} between 5 and {}
clusters'.format(text1,maxcluster))
        def runkm(dft,maxcluster,slicer):
            global kmeansmemberships
            global output
            #kmeansmemberships = pd.DataFrame([df.PSTK]).transpose()
            #kmeansmemberships.head
            columnlist=dft.columns
            init_column=("%s" % (columnlist[0]))
            #print(init_column)
            last_column=("%s" % (columnlist[(len(columnlist)-2)]))
            #print(last_column)
            df1 = dft.loc[0:slicer,init_column:last_column]
            counter = maxcluster
            while counter > 5:
                df1km = KMeans(n_clusters=counter)
                df1km.fit(df1)
                print(counter,'Counter commenced')
                cl_label=("%s clusters with %s"  % (counter,text1))
                df1[(cl_label)]=df1km.labels_
```

```
            print(df1[(cl_label)])
            kmeansmemberships=pd.concat([kmeansmemberships,df1[(cl_label)]],
axis=1, sort=False)
            #Silhouette Coeefficient calculation - the closer to 1 the better the
clustering
            print('\n.....Silhoutte Coeff- the closer to 1 the better the clustering')
            print(".....{}".format(metrics.silhouette_score(df1, df1km.labels_)))
            print('\n')
            #Calinski-Harabaz Index - higher score means better defined clusters
            print('\n.....Calinski-Harabaz Index- higher score means better defined
clusters')
            print(".....{}".format(metrics.calinski_harabaz_score(df1,
df1km.labels_)))
            print('\n')
            # Append outputs to the list
            output.append([text1,'K-
means',counter,'Euclidean',metrics.silhouette_score(df1,
df1km.labels_),metrics.calinski_harabaz_score(df1,
df1km.labels_),'N/A','N/A','N/A'])
            #Davies-Bouldin Index - lower scores mean better defined clusters
            #print('Davies-Bouldin Index')
            #print(davies_bouldin_score(df1, df1km.labels_))
            #print(ta1np.shape)
            #z=z.reshape(51,1)
            #print(z.shape)
```

```
            #print(ta1np)
            #print(z)
            #zz=(np.append(ta1np, z, axis=1))
            #print(zz.shape)
            #print(zz)
            counter=counter-1

        #print(df1)
        #print(df1km.labels_)
        #print(output)
    runkm(k,maxcluster,slicer)
  return(output,kmeansmemberships)
# run the pipeline
pipeline(listkm)
# # C-means clustering

#     cntr : 2d array, size (S, c)
#         Cluster centers.  Data for each center along each feature provided
#         for every cluster (of the `c` requested clusters).
#     u : 2d array, (S, N)
#         Final fuzzy c-partitioned matrix.
#     u0 : 2d array, (S, N)
#         Initial guess at fuzzy c-partitioned matrix (either provided init or
#         random guess used if init was not provided).
#     d : 2d array, (S, N)
```

```
#        Final Euclidian distance matrix.
#    jm : 1d array, length P
#        Objective function history.
#    p : int
#        Number of iterations run.
#    fpc : float
#        Final fuzzy partition coefficient.
# metric list generation
ml1=['cosine','euclidean','chebyshev','cityblock']
ml2='mahalanobis'
cmeansmemberships = pd.DataFrame([df.PSTK]).transpose()
cmeansmemberships2 = pd.DataFrame([df.PSTK]).transpose()
def cmeans(arraylist,metriclist,maxclustersize):
   #output=[]
   ml=metriclist
   global cmeansmemberships
   for i,k in enumerate(listkm):
      maxcluster=5
      k1=k.drop(['PSTK'], axis=1)
      text2=listkmdic.get(listkmstr[i])
      #print(text2)
      #print('Commencing C-means clustering for {} between 25 and {}
clusters'.format(text2,maxclustersize))
      k2=k1.T
      #print('transposition done')
```

```
    # the c-means code
        while maxcluster<maxclustersize:
            for a,b in enumerate(ml):
                met=b
                text3=(maxcluster,'clusters with metric ',b,'for data set ',text2)
                print(text3)
                cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(k2,maxcluster,2,
error=0.005, metric=met, maxiter=10000, init=None)
                print('FPC of',fpc)
                output.append([text3,'C-means',maxcluster,b,'N/A','N/A','N/A',fpc,'N/A'])
                # PERSISTING CLUSTERING MEMBERSHIPS
                #generate column names
                if fpc>0.3:
                    columnnames=[]
                    for y in range(maxcluster):
                        columnnames.append('{} / cluster no {}'.format(text3,y))
                    members=u
                    print(members)
                    members=members.T
                    dfmembers=pd.DataFrame(data = members, columns = columnnames)
                    print(dfmembers)
                    cmeansmemberships=pd.concat([cmeansmemberships,dfmembers],
axis=1, sort=False)
            maxcluster+=1
            print(maxcluster, 'iterated')
```

```
cmeansmemberships.to_csv('cmeansmemberships{}.csv'.format(time.strftime('%Y%
m%d')))
    print("membership print complete")
    return(output,cmeansmemberships)
def cmeanswithmahalanobis(arraylist,metricss,maxclustersize):
    #output=[]
    ml=metricss
    global cmeansmemberships2
    cmeansmemberships2 = pd.DataFrame([df.PSTK]).transpose()
    for i,k in enumerate(listnonstd):
        maxcluster=5
        text2=listnonstddic.get(listnonstdstr[i])
        #print('Commencing C-means clustering for {} between 5 and {}
clusters'.format(text2,maxcluster))
    # data frame transposition
        k1=k.T
    # the c-means code
        while maxcluster<maxclustersize:
            met=metricss
            text3=(maxcluster,'clusters with metric ','mahalanobis',' for data set ',text2)
            cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(k1,maxcluster,2, error=0.005,
metric=met, maxiter=10000, init=None)
            #print(maxcluster,'clusters with an FPC of',fpc,'and distance metric of',met)
            output.append([text2,'C-means',maxcluster,met,'N/A','N/A','N/A',fpc,'N/A'])
```

```
        # PERSISTING CLUSTERING MEMBERSHIPS
            #generate column names
        if fpc>0.3:
            for y in range(maxcluster):
                columnnames.append('{} / cluster no {}'.format(text3,y))
            members=u
            print(members)
            members=members.T
            dfmembers=pd.DataFrame(data = members, columns = columnnames)
            print(dfmembers)
            cmeansmemberships2=pd.concat([cmeansmemberships2,dfmembers],
axis=1, sort=False)
        maxcluster+=1

cmeansmemberships2.to_csv('cmeansmemberships2{}.csv'.format(time.strftime('%Y
%m%d')))
    print("membership print complete")
    return(output,cmeansmemberships2)
cmeans(listkm,ml1,20)

cmeanswithmahalanobis(listnonstd,ml2,35)
# # Affinity propagation
# Metric list
metriclist1=['chebyshev','sqeuclidean','cityblock','minkowski','cosine']
#smlist=['cityblock','cosine']
```

```
afmemberships=pd.DataFrame()
# AP code for non-mahalanobis distances (uses standardised datasets)
def af():
    for i, k in enumerate(listkm):
        text2=listkmdic.get(listkmstr[i])
        # Drop the Primary key
        affinityframe=k.drop(['PSTK'], axis=1)
        for x,y in enumerate(metriclist1):
            if y == 'minkowski':
                # Generate distance matrix for minkowski metric
                print('distance is minkowski')
                pc=np.size(affinityframe,1)
                print("size of frame is:")
                print(pc)
                dst=distance.cdist(affinityframe, affinityframe, 'minkowski', p=pc)
                m1 = np.amin(affinityframe.values)
                # Assign the median to the zero elements
                print("original dst:")
                print(dst[0:2])
                dst[dst == 0] = m1
                dst=dst*-1
                dst1 = pd.DataFrame(dst)
                print('modified dst:')
                print(dst[0:2])
                kd=k.drop(['PSTK'], axis=1)
```

```
    kdd=distance.cdist(kd,kd,y)
    print('k derived dst')
    print(kdd[0:2])
    #
else:
    print('distance is {}'.format(y))
    # Generate distance matrix for non-minkowski metrics
    dst=distance.cdist(affinityframe, affinityframe, y)
    m1 = np.amin(affinityframe.values)
    print("original dst:")
    print(dst[0:2])
    # Assign the median to the zero elements
    dst[dst == 0] = m1
    dst=dst*-1
    dst1 = pd.DataFrame(dst)
    print('modified DST')
    print(dst[0:2])
    kd=k.drop(['PSTK'], axis=1)
    kdd=distance.cdist(kd,kd,y)
    print('k derived dst:')
    print(kdd[0:2])
    #afmemberships[y]=afc.labels_
afc = AffinityPropagation(affinity='precomputed').fit(dst1)
labels=afc.labels_
if metrics.silhouette_score(kdd, labels, metric='precomputed') > 0.2:
```

```
            heading='dataset {} with {} metric:'.format(listkmstr[i],metriclist1[x])
            afmemberships[heading]=afc.labels_
        #k[y]=labels
            print('{} number of clusters extracted'.format(max(labels)))
            print('Silhouette score for dataset {} with {}
metric:'.format(listkmstr[i],metriclist1[x]))
        #if y in smlist:
        #    print(metrics.silhouette_score(affinityframe, labels,
metric=y,sample_size=None, random_state=None))
        #else:
            print(metrics.silhouette_score(kdd, labels, metric='precomputed'))
        #print(metrics.silhouette_score(dst1, labels,'sample_size=None',
'random_state=None'))
        #print('Calinski-Harabaz score for dataset {} with {} metric for {} number of
clusters:'.format(listkmstr[i],metriclist1[x],max(labels)))
        #print(metrics.calinski_harabaz_score(dst1, labels))
            output.append([text2,'Affinity
Propagation',max(labels),y,metrics.silhouette_score(kdd, labels,
metric='precomputed'),metrics.calinski_harabaz_score(dst1,
labels),'N/A','N/A','N/A'])
    afmemberships.to_csv('afmemberships{}.csv'.format(time.strftime('%Y%m%d')))
    return(output,afmemberships)
af()
```

```
print(afmemberships)
# AP code for mahalanobis distances (uses non-standardised datasets)
def afnonstd():
    for i, k in enumerate(listnonstd):
        text2=listnonstddic.get(listnonstdstr[i])
        # Drop the Primary key
        #affinityframe=k.drop(['PSTK'], axis=1)
        affinityframe=k
        y='mahalanobis'
        print('distance is mahalanobis'.format(y))
        # Generate distance matrix for non-minkowski metrics
        dst=distance.cdist(affinityframe, affinityframe, y)
        m1 = np.amin(affinityframe.values)
        # Assign the median to the zero elements
        dst[dst == 0] = m1
        dst=dst*-1
        dst1 = pd.DataFrame(dst)
        #print('moidified DST')
        #print(dst[0:2])
        #print('k derived dst')
        #kd=k.drop(['PSTK'], axis=1)
        kd=k
        kdd=distance.cdist(kd,kd,y)
        #print(kdd[0:2])
        afc = AffinityPropagation(affinity='precomputed').fit(dst1)
```

```
        labels=afc.labels_
        if metrics.silhouette_score(kdd, labels, metric='precomputed') > 0.4:
            heading='dataset {} with {} metric:'.format(listkmstr[i],metriclist1[x])
            afmemberships[heading]=afc.labels_
        k[y]=labels
        print('{} number of clusters extracted'.format(max(labels)))
        print('Silhouette score for dataset {}:'.format(listkmstr[i]))
        #if y in smlist:
        #    print(metrics.silhouette_score(affinityframe, labels,
metric=y,sample_size=None, random_state=None))
        #else:
        print(metrics.silhouette_score(kdd, labels, metric='precomputed'))
        #print(metrics.silhouette_score(dst1, labels,'sample_size=None',
'random_state=None'))
        print('Calinski-Harabaz score for dataset {} for {} number of
clusters:'.format(listkmstr[i],max(labels)))
        print(metrics.calinski_harabaz_score(dst1, labels))
        output.append([text2,'Affinity
Propagation',max(labels),'y',metrics.silhouette_score(kdd, labels,
metric='precomputed'),metrics.calinski_harabaz_score(dst1,
labels),'N/A','N/A','N/A'])
    afmemberships.to_csv('afmemberships{}.csv'.format(time.strftime('%Y%m%d')))
    return(output)
```

```
afnonstd()
# # Generate frame from output list
dfo=pd.DataFrame(output,columns=['InputArray','Algo','n_used','Distance
Function','Silhouette','Calinski-Harabaz Index','Davies Bouldin Score','FPC','GoF-
TBC'])
from IPython.display import display
grid = qgrid.QGridWidget(df=dfo)
display(grid)
dfo.to_csv('dfoutputsummary{}.csv'.format(time.strftime('%Y%m%d')))
cmeansmemberships.to_csv('cmeansmemberships{}.csv'.format(time.strftime('%Y%
m%d')))
kmeansmemberships.to_csv('kmeansmemberships{}.csv'.format(time.strftime('%Y%
m%d')))
cmeansmemberships2.to_csv('cmeansmemberships2{}.csv'.format(time.strftime('%Y
%m%d')))
print(dfo2)
testkm = KMeans(n_clusters=25)
"""
df1km.fit(df1)
print(counter,'Counter commenced')
cl_label=("%s clusters for %s dataset" % (counter,text1))
df1[(cl_label)]=df1km.labels_
kmeansmemberships=pd.concat([kmeansmemberships,df1], axis=1, sort=False)
"""
print(dfss2naPCA)
```

```
testkm=dfss2naPCA
testkmf=testkm.drop(['PSTK'],axis=1)
print(testkmf)
testkmfit = KMeans(n_clusters=25)
testkmfit.fit(testkmf)
testkm[('cl_label')]=testkmfit.labels_
print(testkm)
testkmoutput=testkm.drop(['PC 1','PC 2','PC 3','PC 4','PC 5','PC 6','PC 7','PC 8','PC
9','PC 10','PC 11','PC 12','PC 13','PC 14','PC 15'],axis=1)
print(testkmoutput)
testkmoutput.to_csv('testkmoutput.csv')
listnonstd11=['chebyshev']
#afmemberships=[]
#afmemberships=pd.DataFrame()
#afmemberships=dfnaPCA['PSTK']
#print(afmemberships)
# def afnonstd111():
#     # Drop the Primary key
#     k=dfnaPCA
#     affinityframe=k.drop(['PSTK'], axis=1)
#     for x,y in enumerate(metriclist1):
#         if y == 'minkowski':
#         # Generate distance matrix for minkowski metric
#         #print('distance is minkowski')
```

```
#            pc=np.size(affinityframe,1)
#            #print(pc)
#            dst=distance.cdist(affinityframe, affinityframe, 'minkowski', p=pc)
#            m1 = np.amin(affinityframe.values)
#            # Assign the median to the zero elements
#            dst[dst == 0] = m1
#            dst=dst*-1
#            dst1 = pd.DataFrame(dst)
#            #print('modified dst')
#             #print(dst[0:2])
#               #print('k derived dst')
#            kd=k.drop(['PSTK'], axis=1)
#            kdd=distance.cdist(kd,kd,y)
#               #print(kdd[0:2])
#            labels=afc.labels_
#            afmemberships[y]=afc.labels_
#        else:
#            print('distance is {}'.format(y))
#            # Generate distance matrix for non-minkowski metrics
#            dst=distance.cdist(affinityframe, affinityframe, y)
#            m1 = np.amin(affinityframe.values)
#            # Assign the median to the zero elements
#            dst[dst == 0] = m1
#            dst=dst*-1
#            dst1 = pd.DataFrame(dst)
```

```
#             #print('moidified DST')
#               #print(dst[0:2])
#               #print('k derived dst')
#           kd=k.drop(['PSTK'], axis=1)
#           kdd=distance.cdist(kd,kd,y)
#           #print(kdd[0:2])
#           afc = AffinityPropagation(affinity='precomputed').fit(dst1)
#           labels=afc.labels_
#           #print(labels)
#           afmemberships[y]=afc.labels_
#           #print(k)
#           #print('{} number of clusters extracted'.format(max(labels)))
#           #print('Silhouette score for dataset {} with {}
metric:'.format(listkmstr[i],metriclist1[x]))
#           #if y in smlist:
#           #    print(metrics.silhouette_score(affinityframe, labels,
metric=y,sample_size=None, random_state=None))
#           #else:
#           #print(metrics.silhouette_score(kdd, labels, metric='precomputed'))
#           #print(metrics.silhouette_score(dst1, labels,'sample_size=None',
'random_state=None'))
#           #print('Calinski-Harabaz score for dataset {} with {} metric for {} number
of clusters:'.format(listkmstr[i],metriclist1[x],max(labels)))
#           #print(metrics.calinski_harabaz_score(dst1, labels))
```

```
#            #output.append([text2,'Affinity
Propagation',max(labels),y,metrics.silhouette_score(kdd, labels,
metric='precomputed'),metrics.calinski_harabaz_score(dst1,
labels),'N/A','N/A','N/A'])
#     return(afmemberships)

# # Factor analysis code
#dfss1nax, dfnax
from sklearn.decomposition import FactorAnalysis
transformerfull = FactorAnalysis(n_components=15,svd_method='lapack',
random_state=0)
transformer1 = FactorAnalysis(n_components=7,svd_method='lapack',
random_state=0)

X_transformedfull = transformer.fit_transform(dfnax)
X_transformed1 = transformer.fit_transform(dfss1nax)

#print(X_transformed)
print(X_transformedfull.shape)
print(X_transformed1.shape)
dfX_transformedfull = pd.DataFrame(data=X_transformedfull)
dfX_transformed1 = pd.DataFrame(data=X_transformed1)
listkm999=[dfX_transformedfull,dfX_transformed1]
listkmstr999=['X_transformedfull','X_transformed1']
listkmdic999={
```

```
    "X_transformedfull":"Full dataset factor analysis",
    'X_transformed1':"Subset 1 - traditional box score dataset - factor analysis",
    }
def pipeline999(x):
    for i,k in enumerate(listkm999):
        text1=listkmdic999.get(listkmstr999[i])
        print('Commencing K-means clustering for {} between 5 and {}
clusters'.format(text1,maxcluster))
        def runkm999(dft,maxcluster,slicer):
            global kmeansmemberships
            global output
            #kmeansmemberships = pd.DataFrame([df.PSTK]).transpose()
            #kmeansmemberships.head
            columnlist=dft.columns
            init_column=("%s" % (columnlist[0]))
            #print(init_column)
            last_column=("%s" % (columnlist[(len(columnlist)-2)]))
            #print(last_column)
            df1 = dft.loc[0:slicer,init_column:last_column]
            counter = maxcluster
            while counter > 5:
                df1km = KMeans(n_clusters=counter)
                df1km.fit(df1)
                print(counter,'Counter commenced')
                cl_label=("%s clusters with %s"  % (counter,text1))
```

```
            df1[(cl_label)]=df1km.labels_
            print(df1[(cl_label)])
            kmeansmemberships=pd.concat([kmeansmemberships,df1[(cl_label)]],
axis=1, sort=False)
            #Silhouette Coeefficient calculation - the closer to 1 the better the
clustering
            print('\n.....Silhoutte Coeff- the closer to 1 the better the clustering')
            print(".....{}".format(metrics.silhouette_score(df1, df1km.labels_)))
            print('\n')
            #Calinski-Harabaz Index - higher score means better defined clusters
            print('\n.....Calinski-Harabaz Index- higher score means better defined
clusters')
            print(".....{}".format(metrics.calinski_harabaz_score(df1,
df1km.labels_)))
            print('\n')
            # Append outputs to the list
            output.append([text1,'K-
means',counter,'Euclidean',metrics.silhouette_score(df1,
df1km.labels_),metrics.calinski_harabaz_score(df1,
df1km.labels_),'N/A','N/A','N/A'])
            #Davies-Bouldin Index - lower scores mean better defined clusters
            #print('Davies-Bouldin Index')
            #print(davies_bouldin_score(df1, df1km.labels_))
            #print(ta1np.shape)
            #z=z.reshape(51,1)
```

```
                #print(z.shape)
                #print(ta1np)
                #print(z)
                #zz=(np.append(ta1np, z, axis=1))
                #print(zz.shape)
                #print(zz)
                counter=counter-1

            #print(df1)
            #print(df1km.labels_)
            #print(output)
        runkm999(k,maxcluster,slicer)
    return(output,kmeansmemberships)
pipeline999(listkm999)
output=[]
kmeansmemberships = pd.DataFrame([df.PSTK]).transpose()
```

APPENDIX B: GA ANN CODE: MAIN.PY

```
"""Entry point to evolving the neural network. Start here."""
from __future__ import print_function

from evolver import Evolver

from tqdm import tqdm

import logging
import tensorflow as tf
import sys

# Setup logging.
logging.basicConfig(
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%m/%d/%Y %I:%M:%S %p',
    level=logging.INFO#,
    #filename='log.txt'
)

def train_genomes(genomes, dataset):
    """Train each genome.

    Args:
        networks (list): Current population of genomes
```

```
        dataset (str): Dataset to use for training/evaluating

    """
    logging.info("***train_networks(networks, dataset)***")

    pbar = tqdm(total=len(genomes))

    for genome in genomes:
        genome.train(dataset)
        pbar.update(1)

    pbar.close()

def get_average_accuracy(genomes):
    """Get the average accuracy for a group of networks/genomes.

    Args:
        networks (list): List of networks/genomes

    Returns:
        float: The average accuracy of a population of networks/genomes.

    """
    total_accuracy = 0
```

```
    for genome in genomes:
        total_accuracy += genome.accuracy

    return total_accuracy / len(genomes)

def generate(generations, population, all_possible_genes, dataset):
    """Generate a network with the genetic algorithm.

    Args:
        generations (int): Number of times to evolve the population
        population (int): Number of networks in each generation
        all_possible_genes (dict): Parameter choices for networks
        dataset (str): Dataset to use for training/evaluating

    """
    logging.info("***generate(generations, population, all_possible_genes,
dataset)***")

    evolver = Evolver(all_possible_genes)

    genomes = evolver.create_population(population)

    # Evolve the generation.
    for i in range( generations ):
```

```
    logging.info("***Now in generation %d of %d***" % (i + 1, generations))

    print_genomes(genomes)

    # Train and get accuracy for networks/genomes.
    train_genomes(genomes, dataset)

    # Get the average accuracy for this generation.
    average_accuracy = get_average_accuracy(genomes)

    # Print out the average accuracy each generation.
    logging.info("Generation average: %.2f%%" % (average_accuracy * 100))
    logging.info('-'*80) #----------

    # Evolve, except on the last iteration.
    if i != generations - 1:
        # Evolve!
        genomes = evolver.evolve(genomes)

# Sort our final population according to performance.
genomes = sorted(genomes, key=lambda x: x.accuracy, reverse=True)

# Print out the top 50 networks/genomes.
print_genomes(genomes[:50])
```

```
    saver = tf.train.Saver()
    save_path = saver.save(sess, '/output/model.ckpt')
    print("Model saved in file: %s" % save_path)

def print_genomes(genomes):
    """Print a list of genomes.

    Args:
        genomes (list): The population of networks/genomes

    """
    logging.info('-'*80)

    for genome in genomes:
        genome.print_genome()

def main():
    """Evolve a genome."""
    population = 500 # Number of networks/genomes in each generation.
    #we only need to train the new ones....

    ds = 1

    if(   ds == 1):
        dataset = 'mnist_mlp'
```

```
    elif (ds == 2):
        dataset = 'mnist_cnn'
    elif (ds == 3):
        dataset = 'cifar10_mlp'
    elif (ds == 4):
        dataset = 'cifar10_cnn'
    else:
        dataset = 'mnist_mlp'

    print("***Dataset:", dataset)

    if dataset == 'mnist_cnn':
        generations = 8 # Number of times to evolve the population.
        all_possible_genes = {
            'nb_neurons': [5, 8, 16, 32, 64, 128, 256, 512],
            'nb_layers':  [1, 2, 3, 4 ,5, 6],
            'activation': ['relu', 'softmax', 'elu', 'tanh', 'sigmoid',
'hard_sigmoid','softplus','linear'],
            'optimizer':  ['rmsprop', 'adam', 'sgd', 'adagrad','adadelta', 'adamax', 'nadam']
        }
    elif dataset == 'mnist_mlp':
        generations = 300 # Number of times to evolve the population.
        all_possible_genes = {
            'nb_neurons': [4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 256],
            'nb_layers':  [2, 3, 4, 5],
```

```
        'activation': ['relu', 'elu', 'tanh', 'sigmoid', 'hard_sigmoid','softplus','linear'],
        'optimizer':  ['rmsprop', 'adam', 'sgd', 'adagrad','adadelta', 'adamax', 'nadam']
    }
elif dataset == 'cifar10_mlp':
    generations = 8 # Number of times to evolve the population.
    all_possible_genes = {
        'nb_neurons': [64, 128, 256, 512, 768, 1024],
        'nb_layers':  [1, 2, 3, 4, 5],
        'activation': ['relu', 'elu', 'tanh', 'sigmoid', 'hard_sigmoid','softplus','linear'],
        'optimizer':  ['rmsprop', 'adam', 'sgd', 'adagrad','adadelta', 'adamax', 'nadam']
    }
elif dataset == 'cifar10_cnn':
    generations = 8 # Number of times to evolve the population.
    all_possible_genes = {
        'nb_neurons': [16, 32, 64, 128],
        'nb_layers':  [1, 2, 3, 4, 5],
        'activation': ['relu', 'elu', 'tanh', 'sigmoid', 'hard_sigmoid','softplus','linear'],
        'optimizer':  ['rmsprop', 'adam', 'sgd', 'adagrad','adadelta', 'adamax', 'nadam']
    }
else:
    generations = 8 # Number of times to evolve the population.
    all_possible_genes = {
        'nb_neurons': [64, 128, 256, 512, 768, 1024],
        'nb_layers':  [1, 2, 3, 4, 5],
        'activation': ['relu', 'elu', 'tanh', 'sigmoid', 'hard_sigmoid','softplus','linear'],
```

```
            'optimizer':  ['rmsprop', 'adam', 'sgd', 'adagrad','adadelta', 'adamax', 'nadam']
        }

    # replace nb_neurons with 1 unique value for each layer
    # 6th value reserved for dense layer
    nb_neurons = all_possible_genes['nb_neurons']
    for i in range(1,7):
      all_possible_genes['nb_neurons_' + str(i)] = nb_neurons
    # remove old value from dict
    all_possible_genes.pop('nb_neurons')

    print("***Evolving for %d generations with population size = %d***" %
(generations, population))

    generate(generations, population, all_possible_genes, dataset)

if __name__ == '__main__':
    main()
```

```
"""The genome to be evolved."""

import random
import logging
import hashlib
import copy

from train import train_and_score

class Genome():
    """
    Represents one genome and all relevant utility functions (add, mutate, etc.).
    """

    def __init__( self, all_possible_genes = None, geneparam = {}, u_ID = 0,
mom_ID = 0, dad_ID = 0, gen = 0 ):
        """Initialize a genome.

        Args:
            all_possible_genes (dict): Parameters for the genome, includes:
                gene_nb_neurons_i (list): [64, 128, 256]      for (i=1,...,6)
                gene_nb_layers (list):  [1, 2, 3, 4]
                gene_activation (list): ['relu', 'elu']
                gene_optimizer (list):  ['rmsprop', 'adam']
```

```
        """
        self.accuracy        = 0.0
        self.all_possible_genes = all_possible_genes
        self.geneparam       = geneparam #(dict): represents actual genome parameters
        self.u_ID            = u_ID
        self.parents         = [mom_ID, dad_ID]
        self.generation      = gen

        #hash only makes sense when we have specified the genes
        if not geneparam:
            self.hash = 0
        else:
            self.update_hash()

    def update_hash(self):
        """
        Refesh each genome's unique hash - needs to run after any genome changes.
        """
        genh = str(self.nb_neurons()) + self.geneparam['activation'] \
                + str(self.geneparam['nb_layers']) + self.geneparam['optimizer'] #+
str(self.geneparam['nb_learningrate'])

        self.hash = hashlib.md5(genh.encode("UTF-8")).hexdigest()

        self.accuracy = 0.0
```

```
def set_genes_random(self):
    """Create a random genome."""
    #print("set_genes_random")
    self.parents = [0,0] #very sad - no parents :(

    for key in self.all_possible_genes:
        self.geneparam[key] = random.choice(self.all_possible_genes[key])

    self.update_hash()

def mutate_one_gene(self):
    """Randomly mutate one gene in the genome.

    Args:
        network (dict): The genome parameters to mutate

    Returns:
        (Genome): A randomly mutated genome object

    """
    # Which gene shall we mutate? Choose one of N possible keys/genes.
    gene_to_mutate = random.choice( list(self.all_possible_genes.keys()) )

    # And then let's mutate one of the genes.
```

```
    # Make sure that this actually creates mutation
    current_value    = self.geneparam[gene_to_mutate]
    possible_choices = copy.deepcopy(self.all_possible_genes[gene_to_mutate])

    possible_choices.remove(current_value)

    self.geneparam[gene_to_mutate] = random.choice( possible_choices )

    self.update_hash()

def set_generation(self, generation):
    """needed when a genome is passed on from one generation to the next.
    the id stays the same, but the generation is increased"""

    self.generation = generation
    #logging.info("Setting Generation to %d" % self.generation)

def set_genes_to(self, geneparam, mom_ID, dad_ID):
    """Set genome properties.
    this is used when breeding kids

    Args:
       genome (dict): The genome parameters
    IMPROVE
    """
```

```
        self.parents  = [mom_ID, dad_ID]

        self.geneparam = geneparam

        self.update_hash()

    def train(self, trainingset):
        """Train the genome and record the accuracy.

        Args:
            dataset (str): Name of dataset to use.

        """
        if self.accuracy == 0.0: #don't bother retraining ones we already trained
            self.accuracy = train_and_score(self, trainingset)

    def print_genome(self):
        """Print out a genome."""
        self.print_geneparam()
        logging.info("Acc: %.2f%%" % (self.accuracy * 100))
        logging.info("UniID: %d" % self.u_ID)
        logging.info("Mom and Dad: %d %d" % (self.parents[0], self.parents[1]))
        logging.info("Gen: %d" % self.generation)
        logging.info("Hash: %s" % self.hash)
```

```
    def print_genome_ma(self):
        """Print out a genome."""
        self.print_geneparam()
        logging.info("Acc: %.2f%% UniID: %d Mom and Dad: %d %d Gen: %d" %
(self.accuracy * 100, self.u_ID, self.parents[0], self.parents[1], self.generation))
        logging.info("Hash: %s" % self.hash)

    # print nb_neurons as single list
    def print_geneparam(self):
        g = self.geneparam.copy()
        nb_neurons = self.nb_neurons()
        for i in range(1,7):
          g.pop('nb_neurons_' + str(i))
        # replace individual layer numbers with single list
        g['nb_neurons'] = nb_neurons
        logging.info(g)

    # convert nb_neurons_i at each layer to a single list
    def nb_neurons(self):
      nb_neurons = [None] * 6
      for i in range(0,6):
        nb_neurons[i] = self.geneparam['nb_neurons_' + str(i+1)]
      return nb_neurons
```

APPENDIX D: GA ANN CODE: IDGEN.PY

```
"""Provide unique genome IDs."""

import logging

class IDgen():
    """Generate unique IDs.
    """

    def __init__(self):
        """Keep track of IDs.
        """
        self.currentID  = 0
        self.currentGen = 1

    def get_next_ID(self):

        self.currentID += 1

        return self.currentID

    def increase_Gen(self):

        self.currentGen += 1
```

```
def get_Gen(self):

    return self.currentGen
```

APPENDIX E: GA ANN CODE: EVOLVER.PY

```
"""
Class that holds a genetic algorithm for evolving a network.


Inspiration:


    http://lethain.com/genetic-algorithms-cool-name-damn-simple/
"""
from __future__ import print_function


import random
import logging
import copy


from functools  import reduce
from operator   import add
from genome     import Genome
from idgen      import IDgen
from allgenomes import AllGenomes



class Evolver():
    """Class that implements genetic algorithm."""
```

```
    def __init__(self, all_possible_genes, retain=0.25, random_select=0.1,
mutate_chance=0.1):
        """Create an optimizer.

        Args:
            all_possible_genes (dict): Possible genome parameters
            retain (float): Percentage of population to retain after
                each generation
            random_select (float): Probability of a rejected genome
                remaining in the population
            mutate_chance (float): Probability a genome will be
                randomly mutated

        """

        self.all_possible_genes = all_possible_genes
        self.retain             = retain
        self.random_select      = random_select
        self.mutate_chance      = mutate_chance

        #set the ID gen
        self.ids = IDgen()

    def create_population(self, count):
        """Create a population of random networks.
```

```
        Args:
            count (int): Number of networks to generate, aka the
                size of the population

        Returns:
            (list): Population of network objects

        """
        pop = []

        i = 0

        while i < count:

            # Initialize a new genome.
            genome = Genome( self.all_possible_genes, {}, self.ids.get_next_ID(), 0, 0,
self.ids.get_Gen() )

            # Set it to random parameters.
            genome.set_genes_random()

            if i == 0:
                #this is where we will store all genomes
                self.master = AllGenomes( genome )
```

```
        else:
            # Make sure it is unique....
            while self.master.is_duplicate( genome ):
                genome.mutate_one_gene()

        # Add the genome to our population.
        pop.append(genome)

        # and add to the master list
        if i > 0:
            self.master.add_genome(genome)

        i += 1

    #self.master.print_all_genomes()

    #exit()

    return pop

@staticmethod
def fitness(genome):
    """Return the accuracy, which is our fitness function."""
    return genome.accuracy
```

```
def grade(self, pop):
    """Find average fitness for a population.

    Args:
        pop (list): The population of networks/genome

    Returns:
        (float): The average accuracy of the population

    """
    summed = reduce(add, (self.fitness(genome) for genome in pop))
    return summed / float((len(pop)))

def breed(self, mom, dad):
    """Make two children from parental genes.

    Args:
        mother (dict): genome parameters
        father (dict): genome parameters

    Returns:
        (list): Two network objects

    """
    children = []
```

```
        #where do we recombine? 0, 1, 2, 3, 4... N?
        #with four genes, there are three choices for the recombination
        # ___ * ___ * ___ * ___
        #0 -> no recombination, and N == length of dictionary -> no recombination
        #0 and 4 just (re)create more copies of the parents
        #so the range is always 1 to len(all_possible_genes) - 1
        pcl = len(self.all_possible_genes)

        recomb_loc = random.randint(1,pcl - 1)

        #for _ in range(2): #make _two_ children - could also make more
        child1 = {}
        child2 = {}

        #enforce defined genome order using list
        #keys = ['nb_neurons', 'nb_layers', 'activation', 'optimizer', 'learningrate']
        keys = list(self.all_possible_genes)
        keys = sorted(keys) #paranoia - just to make sure we do not add unintentional
randomization

        #*** CORE RECOMBINATION CODE ****
        for x in range(0, pcl):
            if x < recomb_loc:
                child1[keys[x]] = mom.geneparam[keys[x]]
```

```
            child2[keys[x]] = dad.geneparam[keys[x]]
        else:
            child1[keys[x]] = dad.geneparam[keys[x]]
            child2[keys[x]] = mom.geneparam[keys[x]]

    # Initialize a new genome
    # Set its parameters to those just determined
    # they both have the same mom and dad
    genome1 = Genome( self.all_possible_genes, child1, self.ids.get_next_ID(),
mom.u_ID, dad.u_ID, self.ids.get_Gen() )
    genome2 = Genome( self.all_possible_genes, child2, self.ids.get_next_ID(),
mom.u_ID, dad.u_ID, self.ids.get_Gen() )

    #at this point, there is zero guarantee that the genome is actually unique

    # Randomly mutate one gene
    if self.mutate_chance > random.random():
        genome1.mutate_one_gene()

    if self.mutate_chance > random.random():
        genome2.mutate_one_gene()

    #do we have a unique child or are we just retraining one we already have
anyway?
    while self.master.is_duplicate(genome1):
```

```
            genome1.mutate_one_gene()

        self.master.add_genome(genome1)

        while self.master.is_duplicate(genome2):
            genome2.mutate_one_gene()

        self.master.add_genome(genome2)

        children.append(genome1)
        children.append(genome2)

        return children

    def evolve(self, pop):
        """Evolve a population of genomes.

        Args:
            pop (list): A list of genome parameters

        Returns:
            (list): The evolved population of networks

        """
        #increase generation
```

```
        self.ids.increase_Gen()

        # Get scores for each genome
        graded = [(self.fitness(genome), genome) for genome in pop]

        #and use those scores to fill in the master list
        for genome in pop:
            self.master.set_accuracy(genome)

        # Sort on the scores.
        graded = [x[1] for x in sorted(graded, key=lambda x: x[0], reverse=True)]

        # Get the number we want to keep unchanged for the next cycle.
        retain_length = int(len(graded)*self.retain)

        # In this first step, we keep the 'top' X percent (as defined in self.retain)
        # We will not change them, except we will update the generation
        new_generation = graded[:retain_length]

        # For the lower scoring ones, randomly keep some anyway.
        # This is wasteful, since we _know_ these are bad, so why keep rescoring them
without modification?
        # At least we should mutate them
        for genome in graded[retain_length:]:
            if self.random_select > random.random():
```

```
            gtc = copy.deepcopy(genome)

            while self.master.is_duplicate(gtc):
               gtc.mutate_one_gene()

            gtc.set_generation( self.ids.get_Gen() )
            new_generation.append(gtc)
            self.master.add_genome(gtc)

        # Now find out how many spots we have left to fill.
        ng_length      = len(new_generation)

        desired_length = len(pop) - ng_length

        children       = []

        # Add children, which are bred from pairs of remaining (i.e. very high or lower
scoring) genomes.
        while len(children) < desired_length:

            # Get a random mom and dad, but, need to make sure they are distinct
            parents  = random.sample(range(ng_length-1), k=2)

            i_male   = parents[0]
            i_female = parents[1]
```

```
        male   = new_generation[i_male]
        female = new_generation[i_female]

        # Recombine and mutate
        babies = self.breed(male, female)
        # the babies are guaranteed to be novel

        # Add the children one at a time.
        for baby in babies:
            # Don't grow larger than desired length.
            #if len(children) < desired_length:
            children.append(baby)

    new_generation.extend(children)

    return new_generation
```

```
""" Class that keeps track of all genomes trained so far, and their scores.
    Among other things, ensures that genomes are unique.
"""

import random
import logging

from genome import Genome

class AllGenomes():
    """Store all genomes
    """

    def __init__(self, firstgenome):
        """Initialize
        """

        self.population = []
        self.population.append(firstgenome)

    def add_genome(self, genome):
        """Add the genome to our population.
        """
```

```
    for i in range(0,len(self.population)):
        if (genome.hash == self.population[i].hash):
            logging.info("add_genome() ERROR: hash clash - duplicate genome")
            return False

    self.population.append(genome)

    return True

def set_accuracy(self, genome):
    """Add the genome to our population.
    """

    for i in range(0,len(self.population)):
        if (genome.hash == self.population[i].hash):
            self.population[i].accuracy = genome.accuracy
            return

    logging.info("set_accuracy() ERROR: Genome not found")

def is_duplicate(self, genome):
    """Add the genome to our population.
    """

    for i in range(0,len(self.population)):
```

```
        if (genome.hash == self.population[i].hash):
            return True

    return False

def print_all_genomes(self):
    """Print out a genome.
    """

    for genome in self.population:
        genome.print_genome_ma()
```

APPENDIX G: GA ANN CODE: TRAIN.PY

```
"""
Generic setup of the data sources and the model training.

Based on:
    https://github.com/fchollet/keras/blob/master/examples/mnist_mlp.py
and also on
    https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py

"""

#import keras
from keras.datasets       import mnist, cifar10
from keras.models         import Sequential
from keras.layers         import Dense, Dropout, Flatten
from keras.utils.np_utils import to_categorical
from keras.callbacks      import EarlyStopping, Callback
from keras.layers         import Conv2D, MaxPooling2D
from keras                import backend as K
from keras                import optimizers as koptimizers
from numpy                import genfromtxt
import random
import logging

# Helper: Early stopping.
```

```
early_stopper = EarlyStopping( monitor='val_acc', min_delta=0.0001, patience=2,
verbose=1, mode='auto',restore_best_weights=True )

#patience=5)
#monitor='val_loss',patience=2,verbose=0
#In your case, you can see that your training loss is not dropping - which means you
are learning nothing after each epoch.
#It look like there's nothing to learn in this model, aside from some trivial linear-like
fit or cutoff value.

def get_mnist_mlp():
    """Retrieve the MNIST dataset and process the data."""
    # Set defaults.
    nb_classes  = 2 #dataset dependent
    batch_size  = 128
    epochs      = 100
    input_shape = (11,)
    """ Tweak the source data for NBA exercise"""
    x_train = genfromtxt('Xtrain.csv', delimiter=',')
    y_train = genfromtxt('ytrain.csv', delimiter=',')
    x_test = genfromtxt('Xtest.csv', delimiter=',')
    y_test = genfromtxt('ytest.csv', delimiter=',')
    return (nb_classes, batch_size, input_shape, x_train, x_test, y_train, y_test,
epochs)
```

```
def compile_model_mlp(genome, nb_classes, input_shape):
    """Compile a sequential model.

    Args:
        network (dict): the parameters of the network

    Returns:
        a compiled network.

    """
    # Get our network parameters.
    nb_layers  = genome.geneparam['nb_layers' ]
    nb_neurons = genome.nb_neurons()
    activation = genome.geneparam['activation']
    optimizer  = genome.geneparam['optimizer' ]
    #nb_learningrate  = genome.geneparam['nb_learningrate' ]

    logging.info("Architecture:%s,%s,%s,%d" % (str(nb_neurons), activation,
optimizer, nb_layers))
    #logging.info("Architecture:%s,%s,%s,%s,%d" % (str(nb_neurons), activation,
optimizer, str(nb_learningrate), nb_layers))

    model = Sequential()
    """if optimizer == 'rmsprop':
```

```
        optimizer=koptimizers.rmsprop(lr=nb_learningrate)
    else:
        if optimizer== 'adam':
            optimizer=koptimizers.Adam(lr=nb_learningrate)
        else:
            if optimizer== 'sgd':
                optimizer=koptimizers.Adam(lr=nb_learningrate)
            else:
                if optimizer== 'adagrad':
                    optimizer=koptimizers.adagrad(lr=nb_learningrate)
                else:
                    if optimizer=='adadelta':
                        optimizer=koptimizers.adagrad(lr=nb_learningrate)
                    else:
                        if optimizer=='adamax':
                            optimizer=koptimizers.adagrad(lr=nb_learningrate)
                        else:
                            optimizer==koptimizers.nadam(lr=nb_learningrate)
"""
    # Add each layer.
    for i in range(nb_layers):

        # Need input shape for first layer.
        if i == 0:
```

```
            model.add(Dense(nb_neurons[i], activation=activation,
input_shape=input_shape))
        else:
            model.add(Dense(nb_neurons[i], activation=activation))

        model.add(Dropout(0.2))  # CEM was 0.2 for each layer

    # Output layer.
    model.add(Dense(nb_classes, activation='sigmoid'))

    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy']) # CEM

    return model

class LossHistory(Callback):
    def on_train_begin(self, logs={}):
        self.losses = []

    def on_batch_end(self, batch, logs={}):
        self.losses.append(logs.get('loss'))

def train_and_score(genome, dataset):
    """Train the model, return test loss.
```

```
    Args:
        network (dict): the parameters of the network
        dataset (str): Dataset to use for training/evaluating

    """
    logging.info("Getting Keras datasets")

    if dataset   == 'cifar10_mlp':
        nb_classes, batch_size, input_shape, x_train, x_test, y_train, y_test, epochs =
get_mnist_mlp()
    elif dataset == 'cifar10_cnn':
        nb_classes, batch_size, input_shape, x_train, x_test, y_train, y_test, epochs =
get_mnist_mlp()
    elif dataset == 'mnist_mlp':
        nb_classes, batch_size, input_shape, x_train, x_test, y_train, y_test, epochs =
get_mnist_mlp()
    elif dataset == 'mnist_cnn':
        nb_classes, batch_size, input_shape, x_train, x_test, y_train, y_test, epochs =
get_mnist_mlp()

    logging.info("Compling Keras model")

    if dataset   == 'cifar10_mlp':
        model = compile_model_mlp(genome, nb_classes, input_shape)
```

```
    elif dataset == 'cifar10_cnn':
        model = compile_model_mlp(genome, nb_classes, input_shape)
    elif dataset == 'mnist_mlp':
        model = compile_model_mlp(genome, nb_classes, input_shape)
    elif dataset == 'mnist_cnn':
        model = compile_model_mlp(genome, nb_classes, input_shape)

    history = LossHistory()

    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              # using early stopping so no real limit - don't want to waste time on horrible
architectures
              verbose=2,
              validation_data=(x_test, y_test),
              #callbacks=[history]),
              callbacks=[early_stopper])

    score = model.evaluate(x_test, y_test, verbose=2)

    #persist high-performing models to disk for further analysis and potential
inclusion in committees
    if (score[1] > 0.71):
        filenamee="%saccuracy-ID%s%s" % (score[1],random.random(),'.h5')
```

```
    #print(filenamee)
    model.save(filenamee)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

K.clear_session()
        return score[1]  # 1 is accuracy. 0 is loss.
```

APPENDIX H: GA ANN CODE: PREDICT.PY

```
from keras.utils.np_utils import to_categorical
import pandas as pd
import numpy as np
import keras.models
from keras.models         import Sequential
from keras.layers         import Dense, Dropout, Flatten
from keras.utils.np_utils import to_categorical
from keras.callbacks      import EarlyStopping, Callback
from keras.layers         import Conv2D, MaxPooling2D
from keras                import backend as K
from numpy                import genfromtxt
import sys
import os
import csv
from numpy                import savetxt
#Import Model
#modelfile = input("Please enter filename for the saved model including the
extension: ")



#Load Model
print(os.getcwd())
#with open('newinputs20191105kmeans171.csv') as csvfile:
#    reader = csv.DictReader(csvfile)
```

```
#    for row in reader:
#        print(row)
new_model = keras.models.load_model('0.7533875338753387accuracy-
ID0.5414576312759343.h5')

#Import Prediction Dataset
#predictionfile = raw_input("Please enter filename for the input space: ")
x_predict = genfromtxt('finalyearinputsannrefresh20221015-predictioninput TEAM
LEVEL ONLY.csv', delimiter=',')

#Predict Game Results
new_predictions = new_model.predict(x_predict)
output=new_predictions
print(output)
savetxt('predictions.csv',output,delimiter=',')
```

APPENDIX I: SVM CODE

```
import sklearn
from sklearn.cluster import AffinityPropagation
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
from sklearn.decomposition import PCA
import seaborn as sns; sns.set()
#import skfuzzy as fuzz
#import qgrid
import time
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
import itertools
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.model_selection import StratifiedKFold
%config IPCompleter.greedy=True
print('Library import complete')
#generate source dataframe
df=pd.read_csv('3yearinputTEAMLEVELONLY.csv',index_col='GAMEID',
sep=',',header=0)
df.fillna(0, inplace=True)
dffeatures=df.drop(['homewinflag'], axis=1)
dflabels=df['homewinflag']
nplabels=dflabels.as_matrix(columns=None)
npfeatures=dffeatures.as_matrix(columns=None)
nplabels.astype(int)
npfeatures.astype(int)
X_train, X_test, y_train, y_test = train_test_split(dffeatures, dflabels, test_size=0.25)
from sklearn import svm
clf2 = svm.SVC(gamma=0.1,kernel='linear')
clf2.fit(X_train, y_train)
clf2.score(X_test, y_test, sample_weight=None)
dfpredictions=df
dfpredictions['svmpredict']=clf2.predict(dffeatures)
dfpredictions.to_csv('dfpredictions{}.csv'.format(time.strftime('%Y%m%d')))
clf3 = svm.SVC(gamma=0.1,kernel='sigmoid')
clf3.fit(X_train, y_train)
clf3.score(X_test, y_test, sample_weight=None)
clf4 = svm.SVC(gamma=0.1,kernel='poly')
```

```
clf4.fit(X_train, y_train)

clf4.score(X_test, y_test, sample_weight=None)
```

REFERENCES

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1973*, 420–434. https://doi.org/10.1007/3-540-44503-x_27

Arcidiacono, P., Kinsler, J., & Price, J. (2017). Productivity spillovers in team production: Evidence from professional basketball. *Journal of Labor Economics*, *35*(1), 191–225. https://doi.org/10.1086/687529

Bashuk, M. (2009). Using Cumulative Win Probabilities to Predict NCAA Basketball Performance. *MIT Sloan Sports Conference*, 1–10. http://www.sloansportsconference.com/wp-content/uploads/2012/02/Using-Cumulative-Win-Probabilities-to-Predict-NCAA-Performance-Bashuk.pdf

Beckler, M., Wang, H., & Papamichael, M. (2013). NBA Oracle. *Zuletzt Besucht Am*, *1*(1995), 15213. http://www.mbeckler.org/coursework/2008-2009/10701_report.pdf

Belew, R. K., McInerney, J., & Schraudolph, N. N. (1992). Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In *Artificial Life II* (Vol. 10). http://nic.schraudolph.org/bib2html/b2hd-BelMcISch92.html

Bezdek, J. C., & Bezdek, J. C. (1981). Objective Function Clustering. *Pattern Recognition with Fuzzy Objective Function Algorithms*, 43–93. https://doi.org/10.1007/978-1-4757-0450-1_3

Blaikie, A. D., Abud, G. J., David, J. A., & Pasteur, R. D. (2011). *NFL & NCAA Football Prediction using Artificial Neural Networks*. http://ohio5.openrepository.com/ohio5/handle/2374.DEN/3930

Boulier, B. L. (2003). Predicting the outcomes of National Football League games. *International Journal of Forecasting*, *19*(2), 257–270. https://doi.org/10.1016/S0169-2070(01)00144-3

Bourbousson, J., Sève, C., & McGarry, T. (2010). Space-time coordination dynamics in basketball: Part 1. intra- and inter-couplings among player dyads. *Journal of Sports Sciences*, *28*(3), 339–347. https://doi.org/10.1080/02640410903503632

Bunker, R. P., & Thabtah, F. (2019). A machine learning framework for sport result prediction. *Applied Computing and Informatics*, *15*(1), 27–33. https://doi.org/10.1016/j.aci.2017.09.005

Cao, C. (2012). Sports data mining technology used in basketball outcome prediction. *Dublin Institute of Technology*, 1–86. http://arrow.dit.ie/scschcomdis/39%5Cnhttp://arrow.dit.ie/scschcomdis/39/

Cervone, D., Bornn, L., & Goldsberry, K. (2016). NBA Court Realty Intro: the Basketball Court is a Real Estate Market. *MIT Sloan Sports Analytics Conference*, *1*, 1–8.

Chan, T. C. Y., Cho, J. A., & Novati, D. C. (2012). Quantifying the contribution of NHL player types to team performance. *Interfaces*, *42*(2), 131–145. https://doi.org/10.1287/inte.1110.0612

Cherkassky, V., & Ma, Y. (2004). Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, *17*(1), 113–126. https://doi.org/10.1016/S0893-6080(03)00169-2

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Croft, H., Lamb, P., & Middlemas, S. (2015). The application of self-organising maps to performance analysis data in rugby union. *International Journal of Performance Analysis in Sport*, *15*(3), 1037–1046. https://doi.org/10.1080/24748668.2015.11868849

David, J. A., Pasteur, R. D., Ahmad, M. S., & Janning, M. C. (2011). NFL prediction using committees of artificial neural networks. *Journal of Quantitative Analysis in Sports*, *7*(2). https://doi.org/10.2202/1559-0410.1327

Dežman, B., Trninić, S., & Dizdar, D. (2001). Expert model of decision-making system for efficient orientation of basketball players to positions and roles in the game -Empirical verification. *Collegium Antropologicum*, *25*(1), 141–152.

Dong C. Liu, & Jorge Nocedal. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, *45*, 503–528. https://link.springer.com/content/pdf/10.1007%2FBF01589116.pdf

Duarte, R., Araújo, D., Correia, V., & Davids, K. (2012). Sports Teams as Superorganisms. *Sports Medicine*, *42*(8), 1. https://doi.org/10.2165/11632450-000000000-00000

Fewell, J. H., Armbruster, D., Ingraham, J., Petersen, A., & Waters, J. S. (2012). Basketball teams as strategic networks. *PloS One*, *7*(11). https://doi.org/10.1371/journal.pone.0047445

Flitman, A. M. (2006). Towards probabilistic footy tipping: A hybrid approach utilising genetically defined neural networks and linear programming. *Computers and Operations Research*, *33*(7), 2003–2022.

https://doi.org/10.1016/j.cor.2004.09.032

Franks, A., Miller, A., Bornn, L., & Goldsberry, K. (2015). Counterpoints : Advanced Defensive Metrics for NBA Basketball. *MIT Sloan Sports Analytics Conference*, 1–8.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972–976. https://doi.org/10.1126/science.1136800

García, J., IbÁñez, S. J., Cañadas, M., & Antúnez, A. (2013). Complex system theory in team sports. example in 5 on 5 basketball contest. *Revista de Psicologia Del Deporte*, *22*(1), 209–213.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *14th International Conference on Artificial Intelligence and Statistics*, *15*(7), 315–323. https://doi.org/10.1002/ecs2.1832

Goldsberry, K., & Weiss, E. (2013). The Dwight Effect : A New Ensemble of Interior Defense Analytics for the NBA. *MIT Sloan Sports Analytics Conference*, 1–11.

Grossberg, S. (1988). Nonlinear Neural Networks : Principles , Mechanisms , and Architectures. *Neural Networks*, *1*, 17–61.

Halevy, N., Chou, E. Y., Galinsky, A. D., & Murnighan, J. K. (2012). When Hierarchy Wins: Evidence From the National Basketball Association. *Social Psychological and Personality Science*, *3*(4), 398–406. https://doi.org/10.1177/1948550611424225

Herold, M., Goes, F., Nopp, S., Bauer, P., Thompson, C., & Meyer, T. (2019). Machine learning in men's professional football: Current applications and future directions for improving attacking play. *International Journal of Sports Science*

and Coaching, *14*(6), 798–817. https://doi.org/10.1177/1747954119879350

Holland, J. H. (1992). Genetic Algorithms. *Scientific American*, *267*(1), 66–73.

Hore, S., & Bhattacharya, T. (2018). A Machine Learning Based Approach Towards Building a SustainabilityModel for NBA Players. *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, *Icicct*, 1690–1694. https://doi.org/10.1109/ICICCT.2018.8473102

Hu, F., & Zidek, J. V. (2004). *Forecasting NBA basketball playoff outcomes using the weighted likelihood*. *45*, 385–395. https://doi.org/10.1214/lnms/1196285406

Huang, K. Y., & Chang, W. L. (2010). A neural network method for prediction of 2006 World Cup Football Game. *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN.2010.5596458

Kahneman, D., & Frederick, S. (2012). Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In *Heuristics and Biases* (Issue September). https://doi.org/10.1017/cbo9780511808098.004

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Kuehn, J. (2017). Accounting for Complementary Skill Sets: Evaluating Individual Marginal Value To a Team in the National Basketball Association. *Economic Inquiry*, *55*(3), 1556–1578. https://doi.org/10.1111/ecin.12451

Lames, M., & McGarry, T. (2007). On the search for reliable performance indicators in game sports. *International Journal of Performance Analysis in Sport*, *7*(1), 62–79. https://doi.org/10.1080/24748668.2007.11868388

Leung, C. K., & Joseph, K. W. (2014). Sports data mining: Predicting results for the college football games. *Procedia Computer Science*, *35*(C), 710–719. https://doi.org/10.1016/j.procs.2014.08.153

Liphardt, J., Downs, G., & Tiwari, S. (2019). *DeepEvolve*. https://github.com/jliphard/DeepEvolve

Loeffelholz, B., Bednar, E., & Bauer, K. W. (2009). Predicting NBA Games Using Neural Networks. *Journal of Quantitative Analysis in Sports*, *5*(1). https://doi.org/10.2202/1559-0410.1156

Lutz, D. (2012). A Cluster Analysis of NBA Players. *MIT Sloan Sports Analytics Conf*, 1–8. http://www.sloansportsconference.com/wp-content/uploads/2012/02/44-Lutz_cluster_analysis_NBA.pdf

Maheswaran, R., Chang, Y., Su, J., Kwok, S., Levy, T., Wexler, A., & Hollingsworth, N. (2014). The Three Dimensions of Rebounding. *MIT Sloan Sports Analytics Conference*, 1–7.

Martínez, J. A., & Martínez, L. (2011). A stakeholder assessment of basketball player evaluation metrics. *Journal of Human Sport and Exercise*, *6*(1), 153–183. https://doi.org/10.4100/jhse.2011.61.17

Martinez, R. (2017). Complex systems: Theory and applications. *Complex Systems: Theory and Applications*, 1–151.

Martins, R. G., Martins, A. S., Neves, L. A., Lima, L. V., Flores, E. L., & do Nascimento, M. Z. (2017). Exploring polynomial classifier to predict match results in football championships. *Expert Systems with Applications*, *83*, 79–93. https://doi.org/10.1016/j.eswa.2017.04.040

McCabe, A., & Trevathan, J. (2008). Artificial intelligence in sports prediction.

Proceedings - International Conference on Information Technology: New Generations, ITNG 2008, 1194–1197. https://doi.org/10.1109/ITNG.2008.203

Mertz, J., Hoover, D. L., Burke, J. M., Bellar, D., Jones, L. M., Leitzelar, B., & Judge, L. W. (2016). Ranking the greatest NBA players: A sport metrics analysis. *International Journal of Performance Analysis in Sport*, *16*(3), 737–759. https://doi.org/10.1080/24748668.2016.11868925

Miljković, D., Gajić, L., Kovačević, A., & Konjović, Z. (2010). The use of data mining for basketball matches outcomes prediction. *SIISY 2010 - 8th IEEE International Symposium on Intelligent Systems and Informatics*, 309–312. https://doi.org/10.1109/SISY.2010.5647440

Miller, A., Bomn, L., Adams, R., & Goldsberry, K. (2014). Factorized point process intensities: A spatial analysis of professional basketball. *31st International Conference on Machine Learning, ICML 2014*, *1*, 398–414.

Min, B., Kim, J., Choe, C., Eom, H., & (Bob) McKay, R. I. (2008). A compound framework for sports results prediction: A football case study. *Knowledge-Based Systems*, *21*(7), 551–562. https://doi.org/10.1016/j.knosys.2008.03.016

Nakhjavani, A., Mollaverdi, N., & Rafiei, F. M. (2014). *Football Match Results Prediction Using Artificial Neural Networks; The Case of Iran Pro League*. *1*(4), 230–249.

Nevill, A. M., & Holder, R. L. (1999). Home Advantage in Sport. *Sports Medicine*, *28*(4), 221–236. https://doi.org/10.2165/00007256-199928040-00001

Oberhofer, H., & Schwinner, M. (2017). Do Individual Salaries Depend on the Performance of the Peers? Prototype Heuristic and Wage Bargaining in the NBA. *WIFO Working Papers*, *534*.

Oh, M., Keshri, S., & Iyengar, G. (2015). Graphical Model for Baskeball Match Simulation. *MIT Sloan Sports Analytics Conference*, 1–8.

Orendorff, D., & Johnson, T. (2007). First-Order Probabilistic Models for Predicting the Winners of Professional Basketball Games. *Working Paper*, *January 2008*. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:First-Order+Probabilistic+Models+for+Predicting+the+Winners+of+Professional+Basketball+Games#0

Passos, P., Davids, K., Araújo, D., Paz, N., Minguéns, J., & Mendes, J. (2011). Networks as a novel tool for studying team ball sports as complex social systems. *Journal of Science and Medicine in Sport*, *14*(2), 170–176. https://doi.org/10.1016/j.jsams.2010.10.459

Pedregosa, F., Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., & Mueller, A. (2015). Scikit-learn. *GetMobile: Mobile Computing and Communications*, *19*(1), 29–33. https://doi.org/10.1145/2786984.2786995

Piette, J., Pham, L., & Anand, S. (2011). Evaluating basketball player performance via statistical network modeling. *MIT Sloan Sports Analytics Conference*, *June*, 1–11.

Pion, J., Segers, V., Stautemas, J., Boone, J., Lenoir, M., & Bourgois, J. G. (2018). Position-specific performance profiles, using predictive classification models in senior basketball. *International Journal of Sports Science and Coaching*, *13*(6), 1072–1080. https://doi.org/10.1177/1747954118765054

Price, J., Soebbing, B. P., Berri, D., & Humphreys, B. R. (2010). Tournament incentives, league policy, and nba team performance revisited. *Journal of Sports Economics*, *11*(2), 117–135. https://doi.org/10.1177/1527002510363103

Radovanovic, S., Radojicic, M., Jeremic, V., & Savic, G. (2013). A Novel Approach in Evaluating Efficiency of Basketball Players. *Management - Journal for Theory and Practice of Management*, *18*(67), 37–46. https://doi.org/10.7595/management.fon.2013.0012

Rangel, W., Ugrinowitsch, C., & Lamas, L. (2019). Basketball players' versatility: Assessing the diversity of tactical roles. *International Journal of Sports Science and Coaching*, *14*(4), 552–561. https://doi.org/10.1177/1747954119859683

Ross, T. J. (2010). *Fuzzy Logic With Engineering Applications*. Wiley.

Safir, J. (2015). *How Analytics , Big Data , and Technology Have Impacted Basketball ' s Quest to Maximize Efficiency and Optimization*.

Sahu, L., & Mohan, B. R. (2015). An improved K-means algorithm using modified cosine distance measure for document clustering using Mahout with Hadoop. *9th International Conference on Industrial and Information Systems, ICIIS 2014*. https://doi.org/10.1109/ICIINFS.2014.7036661

Sampaio, J., Drinkwater, E. J., & Leite, N. M. (2010). Effects of season period, team quality, and playing time on basketball players' game-related statistics. *European Journal of Sport Science*, *10*(2), 141–149. https://doi.org/10.1080/17461390903311935

Schumaker, R. P., Solieman, O. K., & Chen, H. (2010). Sports Data Mining. In *Springer* (Vol. 26). https://doi.org/10.1007/978-1-4419-6730-5_13

Soebbing, B. P., & Humphreys, B. R. (2013). Do gamblers think that teams tank? Evidence from the NBA. *Contemporary Economic Policy*, *31*(2), 301–313. https://doi.org/10.1111/j.1465-7287.2011.00298.x

Stiroh, K. J. (2007). Playing for keeps: Pay and performance in the NBA. *Economic*

Inquiry, *45*(1), 145–161. https://doi.org/10.1111/j.1465-7295.2006.00004.x

Tarlow, J. (2012). Experience and winning in the National Basketball Association. *MIT Sloan Sports Analytics Conference*, 1–11.

Tax, N., & Joustra, Y. (2015). Predicting The Dutch Football Competition Using Public Data: A Machine Learning Approach. *Transactions on Knowledge and Data Engineering*, *10*(SEPTEMBER), 1–13. https://doi.org/10.13140/RG.2.1.1383.4729

Travassos, B., Davids, K., Araújo, D., & Esteves, P. T. (2013). Performance analysis in team sports: Advances from an ecological dynamics approach. *International Journal of Performance Analysis in Sport*, *13*(1), 83–95. https://doi.org/10.1080/24748668.2013.11868633

Vadruccio, L. S. (2018). Ranking methods for data analytics on players ' performance in basketball games. In *Statistiche, Scienze Magistrale, Laurea*.

Vafaie, H., & De Jong, K. (1992). Genetic algorithms as a tool for feature selection in machine learning. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, *1992-Novem*(November), 200–203. https://doi.org/10.1109/TAI.1992.246402

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Walters, C., & Williams, T. (2012). To Tank or Not to Tank: Evidence from the

NBA. *MIT Sloan Sports Analytics Conference 2012 March 2-3, 2012, Boston, MA, USA*, *2010*, 1–11.

Wang, W., & Zhang, Y. (2007). On fuzzy cluster validity indices. *Fuzzy Sets and Systems*, *158*(19), 2095–2117. https://doi.org/10.1016/j.fss.2007.03.004

Werner, E. E., & Peacor, S. D. (2003). A Review of Trait-Mediated Indirect Interactions in Ecological Communities Introduction and the Conceptual Problem. *Special Feature Ecology*, *84*(5), 1083–1100.

Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, *14*(3), 347–361. https://doi.org/10.1016/0167-8191(90)90086-O

Yang, J., & Lu, C.-H. (2012). Predicting NBA championship by learning from history data. *... Intelligence and Machine Learning for …*, 1–4. http://www.contrib.andrew.cmu.edu/~jackiey/resources/NBAchamp/nba_champ_predict.pdf

Zhang, S., Lorenzo, A., Gómez, M. A., Mateus, N., Gonçalves, B., & Sampaio, J. (2018). Clustering performances in the NBA according to players' anthropometric attributes and playing experience. *Journal of Sports Sciences*, *36*(22), 2511–2520. https://doi.org/10.1080/02640414.2018.1466493