An example of a thesis or dissertation on the subject of your degree

by

Yejia Liu

B.Sc., South China University of Technology, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Spring 2018

Approval

Name:	**Yejia Liu**
Degree:	**Master of Science (Computing Science)**
Title:	**An example of a thesis or dissertation on the subject of your degree**
Examining Committee:	**Chair:** Pamela Isely Professor
	Oliver Schulte Senior Supervisor Professor
	Tim Swartz Supervisor Professor
Date Defended:	**April 15, 2018**

Abstract

Drafting players is crucial for a team's success. We describe a data-driven interpretable approach for assessing prospects in the National Hockey League and National Basketball Association. Previous approaches have built a predictive model based on player features, or derived performance predictions from comparable players. Our work develops model tree learning, which incorporates strengths of both model-based and cohort-based approaches. A model tree partitions the feature space according to the values or learned thresholds of features. Each leaf node in the tree defines a group of players, with its own regression model. Compared to a single model, the model tree forms an ensemble that increases predictive power. Compared to cohort-based approaches, the groups of comparables are discovered from the data, without requiring a similarity metric. The model tree shows better predictive performance than the actual draft order. It can also be used to highlight strongest points of players.

Keywords: player ranking; Logistic Model Tree; M5 regression tree; National Hockey League; National Basketball Association; Spearman rank correlation

Dedication

This is an optional page.

Acknowledgements

This is an optional page.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

Drafting players is one of the most important tasks in any sports to order to build a successful team. This process can take millions of dollars and thousands of man hours. In this thesis, we focus on the draft of two most well-known leagues, National Hockey League(NHL) and National Basketball Association(NBA). To draft prospects, the team often relies heavily on scouts, who may only be able to watch a player a handful of times a season. Another relatively inexpensive way to draft talents is through the Entry Draft, where players who recently become eligible to play in a league are allocated to a team. The entry draft of them both use lottery system to determine which team gets the top picks, so every team is supposed to has a chance to sign a superstar. In this system, the best player is expected to have the first draft pick, the second best have the second draft pick, and so forth. However, the history has shown there are many misfires in the draft pick. In 2008 NHL entry draft, Nikita Filatov gained the 6th overall pick, taken before the well-known Erik Karlsson (No.15), but only played 53 games and scored 6 goals in NHL. In NBA draft, the most notorious pick belongs to the team Portland Trail Blazers, who chose Sam Bowie over Michael Jordan in 1984. To find a more effective and economic way to access draftees, many sport experts statisticians turn to data-driven methods. In this thesis, we consider predicting player future success in NHL and NBA based on datasets from junior leagues (colleges in NBA), then ranking them with prediction results, with the purpose of supporting draft decisions.

Previous work for analyzing NHL/NBA draft datasets mainly include regression approach or similarity-based approach. Regression approaches build a predictive model that takes as input a set of player features, such as demographics metrics(age, height, weight etc.) and pre-draft performance metrics (goals scored, plus-minus, shoots, minutes player etc.), and output a predicted success metric (number of games played for NHL, player efficiency rating(PER) in NBA) [6, 4]. Cohort-based approaches divide players into groups of comparables and predict future success based on the player cohort. For example, the PCS model [32] clusters ice hockey players according to age, height, and scoring rates. One advantage of the cohort model is that predictions can be explained by reference to similar known players, which many domain experts find intuitive. Thus, many commercial sports analytic systems, such as Sony's Hawk-Eye system, have been developed to identify groups of comparables for each player. Yale university also has built a NBA player clustering system to classify players according to their play style and career performance. *(http://sports.sites.yale.edu/clustering-nba-players)*.

In this thesis, we apply our tree model to the pre-draft data that achieves the best of both approaches, regression-based and similarity-based [9, 18]. Each node in the tree defines a yes or no question until a leaf is reached. Based on answers to these questions, each player is allocated to a group corresponding to a leaf. In each leaf node, a regression model is built. Figure 1 shows an example model tree. Compared to a single regression model, the tree defines an ensemble of regression models, based on non-linear thresholds. This increases the expressive power and predictive accuracy of the model. The tree also represents complex interactions between player features and player groups. For example, if the data indicates that players from different junior leagues are sufficiently different to warrant building distinct models, the tree can introduce a split to distinguish different leagues. While compared to a similarity-based model, tree construction learns groups of players from the data, without requiring the analyst to specify a similarity metric. It selects splits that increase predictive accuracy. The learned distinctions between the groups are guaranteed to be predicatively relevant to future national league success. Also, the tree models create a model for each group, which allows to differentiate players from the same group.

More specifically, in the NHL draft, only about half of the drafted prospects finally played a game in NHL [31], which brings up a zero-inflation problem that limits the predictive power of linear regression. Thus, we apply logistic regression to predict whether a player will play at least one game in the NHL. We learn a logistic regression model tree, and rank players by the probability that the logistic regression model tree assigns to them playing at least one game. Intuitively, if we can be confident that a player will play at least one NHL game, we can also expect the player to play many NHL games. While in NBA draft, a more intuitive approach, linear regression tree, is built since there is no such zero inflation issue.

Following [6, 4], we evaluate the model trees ranking results by comparing it to ranking players by their future success, measured as the number of career games they played after 7 years for NHL players, player efficiency rating(PER) for NBA players. We show in case studies that the feature weights learned from the data can be used to explain the ranking in terms of which player features contribute the most to an above-average ranking. In this way the model tree can be used to highlight exceptional features of a player for scouts and teams to take into account in their evaluation.

Thesis Outline. We first review background in NHL/NBA draft and related work in model trees. Then we describe and carry out some statistic analysis of our datasets. After data exploration, the construction of model tree is presented. In the Results part, the

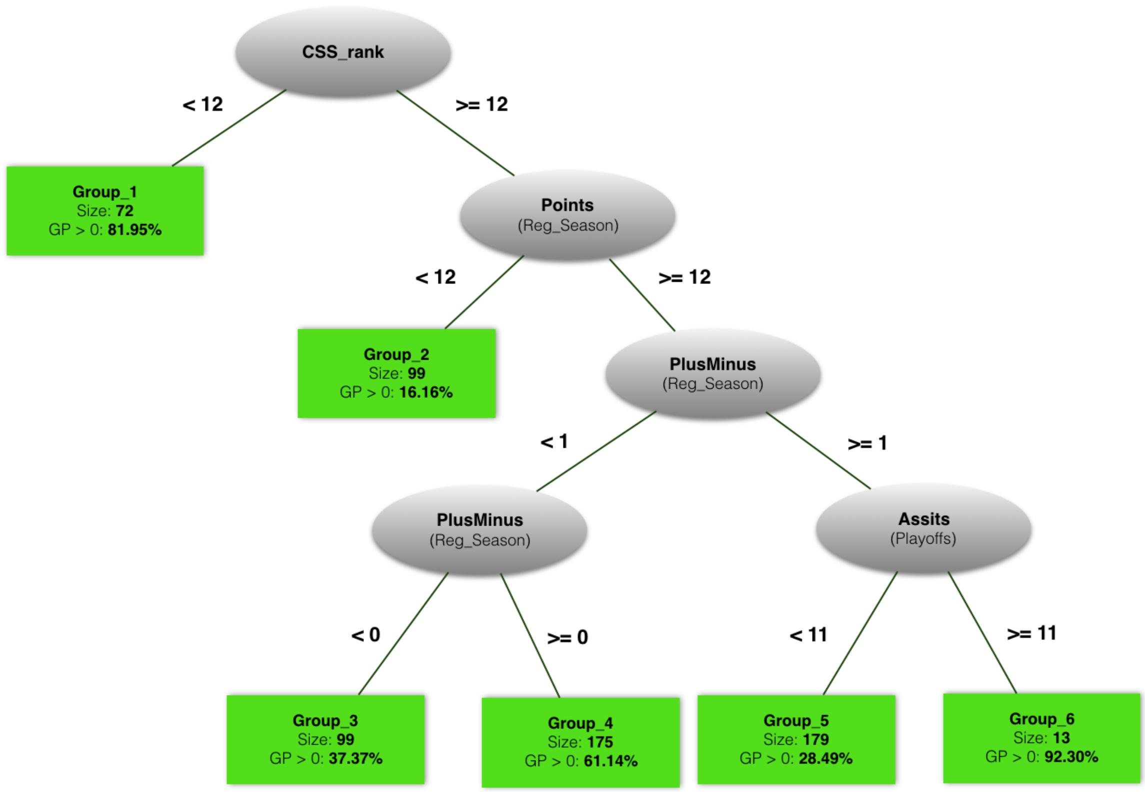

Figure 1.1: Logistic Regression Model Trees for the $2004, 2005, 2006$ cohort in NHL. The tree was built using the LogitBoost algorithm implemented in the LMT package of the Weka Program [8, 11].

rank correlations are reported to evaluate predictive accuracy. Case studies give examples of strong players in different groups and show how the model can be used to highlight exceptional player strongest points.

Chapter 2

Background, Literature Review and Problem Formulation

For different data types, there are different approaches for player ranking. With play-by-play datasets, Markov models have been widely used to analyze player performance [6, 15]. In [28], the NHL player ability ratings are calculated by modeling team scoring rate as semi-Markov process, with hazard functions depending on players on the ice. Bornn proposes a Pointwise function to predict the number of points that a NBA player is expected to score by the end of an action [3]. For data that records the presence of players when a goal is scored, regression models have been applied to extend the classic wins-contribution metrics. For example, Macdonald develops two weighted least squares regression models to evaluate a NHL player's effect on team success in scoring and goals, independent of the opponents [20]. While in NBA, Sill enhances the traditional plus-minus by combing it with ridge regression to produce more accurate results [26]. In our work, we utilize player statistics that aggregates player pre-draft performance into a single set of numbers. While these datasets are less informative than play-by-play statistics, they are easier to obtain, interpret and process.

2.1 Previous Models

Regression Approaches. Wilson use predictive models (Generalized Linear Model, Artificial Neural Network, Support Vector Machine and LOESS) to predict whether a drafted NHL player can play more than 160 games after 7 career years. In his work, the pre-draft statistics and the first four season NHL performance statistics are both used [33]. This enlightens us to look into standalone pre-draft NHL datasets with the purpose of predicting whether a drafted player can play at least one game or not at NHL. Since based on our analysis, almost half of the drafted players will not play a game in NHL [31]. The closest predecessor to our NHL work is due to Schuckers [6], who utilizes the pre-draft datasets to predict future NHL game counts by using a single generalized additive model. His results show strong correlation between player career performance and pre-draft statistics. While in the NBA area, regression approaches have also been widely used to analyze player performance. Coates and Oguntimein examines the relationship between the college metrics and the analogues metrics of NBA productivity through least square regression. Their results show some college statistics are significant in predicting NBA statistics, such as the college scoring and reboundings, which do well in predicting NBA minutes played [5]. In this thesis, we mainly follow Greene's work [4], who build a linear regression model to quantify the likelihood of a drafted college player having a successful NBA career. The inputs of his model are quite extensive, including the player draft picks, college statistics and physical qualities, adjusted by rookie year stats. His work presents a better predictive performance than the actual NBA draft when it comes to the top 100 prospects going to the draft.

Similarity-Based Approaches assume a similarity metric and group similar players to predict performance. A sophisticated example from baseball is the nearest neighbour analysis in the PECOTA system [27]. In the ice hockey field, the Prospect Cohort Success (PCS) model [32], defines cohorts of draftees based on age, height, and scoring rates. While in basketball, many clustering approaches focus on define a player appropriate roles or positions. In Lutz's work [19], NBA players are clustered to several types like Combo Guards, Floor Spacers and Elite Bigs. They are grouped based on games played, minutes played per game, assists, turnover rate, rebound, steals and blocks. His results have shown different type of players can effect the team wining. Yale University also has developed a NBA clustering system to cluster players from season 2011-2012 to season 2015-2016 through hierarchical clustering methodology with their season performance statistics as inputs(`http://sports.sites.yale.edu/clustering-nba-players`). Our model tree learning provides an automatic method for identifying cohorts together with predictive validity. *We refer to grouping results as groups to avoid confusion with cohorts or clusters used in [27, 19].* Because tree learning is computationally efficient, our model tree is able to take into account a broad set of features. Also, it provides a separate predictive model for each group that assigns group-specific weights to different features. In contrast, [32] and [19] make the same prediction for all players in the same cohort. So far, PCS has been applied to predict whether a player will score more than 200 games career total. Tree learning can easily be modified to make predictions for any game count threshold.

2.2 Explanation of Career Success Metrics

The growing business of professional sports has resulted in an increasing demand for effective metrics to quantify a player contribution to a team success. Broadly the crude summary statistics to compare players are three types: goal-based metrics, shot-based metrics and assists.

In this section, we discuss the success metrics used in NHL and NBA to evaluate a player career performance. Among those success metrics, games played for NHL and player efficiency rating(PER) for NBA are chosen as response variables in our model trees.

2.2.1 NHL Player Evaluation Metrics

Goal scoring is an infrequent event in ice hockey compared to other high-scoring sports like basketball. In the NHL there are approximately 10 shots taken for scoring 1 goal. This higher number of shot events leads to measurements focusing on shots taken and shots allowed. In the work of Thomas [30], shots are assumed to be the proxies for zone possession time, where shots are indeed considered as a proxy for team success. However, according to the analytic report from Found, the goal-based metrics (e.g., relative plus-minus/minute of ice time) always outperforms the shot-based metrics (e.g., relative Corsi/minute of ice time), when it comes to access an individual player contributions to the team winning percentage [7]. Thus, it's natural for many sports analysts and statisticians to think of new measuring models which take several generally used success metrics into consideration, with the goal of evaluating players more completely.

Win Shares [12], first created in the world of baseball, are now also widely used in other sports to evaluate player performance. Enlightened by the Win Shares system, Hockey-Reference has built Point Shares system, where one points is equivalent to one point share, and the player contribution is calculated by marginal goals, points and time on ice `https://www.hockey-reference.com/about/point_shares.html`. Below is the main formula in Point Share system for skaters:

$$\text{Skaters Point Shares} = (\text{marginal goals}) \;/\; (\text{marginal goals per point})$$

where marginal goals = (goals created) - (7 / 12) * (time on ice) * ((goals created by forwards or defensemen) / (time on ice for forwards or defensemen)) and marginal goals per point = (league goals) / (league points). This Point System applies commonsense methods to calculating point shares and has been proven to have lower average absolute error in comparison with team's point total on NHL datasets from season $1917-18$ to $2009-10$. However, the usage of magical number in this system may reduce its generality to other seasons datasets. The Total Hockey Rating(ThoR) proposed by Schucker and Curro has gone beyond simple counting statistics, where ridge regression models has been adopted. According to [24], ThoR is a comprehensive rating model which accounts for the impact of where a shift starts, and also non-shooting events including turnovers and hits that occurs when a player is on the ice. The contribution of these actions is then quantified by the probability that wether it leads to a goal for the player's team or not. ThoR has been applied to all the $2010-2011$ and $2011-2012$ NHL ice events, which produced convincing rating lists for both defensemen and forwards among these seasons. Nevertheless, ThoR and other regression models usually require large sets of data, and are often computationally expensive.

Recently, there have also been a lot of studies about Wins Above Replacement(WAR), which mainly uses aggregate and count data. In the report from Thomas and Ventura [29], the shot rates, shot quality(likelihood of a shot becoming a goal), penalty rates and game states have been accounted, resulting in a scalable statistic model.

In this thesis, we uses the total number of games played in a player's first seven years after they have been drafted, following Schucker's work [6], since teams have the rights to players for at least seven years after they are drafted. Compared to other single metrics like plus-minus or complex models like WAR, games played is more intuitive and easier to interpret. Also, games played represents the usage rate of a player inherently.

2.2.2 NBA Player Evaluation Metrics

Basketball, as one of the most popular sports in the world, has been well studies with respect to success metrics to evaluate players. Some of the most intriguing and famous ones include Player Efficiency Rating(PER)(`www.basketball-reference.com/about/per.html`) and Win Shares(WS)(`www.basketball-reference.com/about/ws.html`), which are mainly discussed in this section.

Player Efficiency Rating

The player efficiency rating(PER), created by John Hollinger, takes nearly every aspect of a player contribution into consideration. It encompasses player almost every accomplishment, such as field goals, free throws, 3-pointers, assists, rebounds, blocks and steals. Meanwhile, the negative results, such as missed shots, turnover and personal fouls are also accounted in the rating system. Compared to the traditional success metrics like wins, which highly depends on opportunities created by a player's teammates, PER aims to measuring a single player per-minute performance. In addition, PER is usually adjusted by minutes played ad game pace. Because Hollinger notes that a player's opportunities to accumulate statistics are dependent on the number of minutes played as well as the pace of the game.

The average league PER is always 15, which allows for comparing players across seasons. It has a rough scale which demonstrates the productivity of a player in a given year, listed in Table 2.1. This table provides a good guide to access a player performance over his career. For example, Michael Jordan is widely recognized as one of the best player in NBA and PER supports this claim. He currently has one of the highest career PER 27.91. There are only about 60 players in the history of the NBA having a career PER above 20.

The calculation of PER is as follows:

$$PER = uPER \times \frac{lgPace}{tmPace} \times \frac{15}{lguPER}$$

where $uPER$ is the unadjusted PER, calculated using a large number of variables, including points, rebounds, assists, field goals, free throws, turnovers, and three pointers, as well as team and league statistic.

A Year for the Ages	35.0+
Runaway MVP Candidate	30.0-35.0
Strong MVP Candidate	27.5-30.0
Weak MVP Candidate	25.0-27.5
Definite All-Star	22.5-25.0
Borderline All-Star	20.0-22.5
Second offensive option	18.0-20.0
Third offensive option	16.5-18.0
Slightly above-average player	15.0-16.5
Rotation player	13.0-15.0
Non-rotation player	11.0-13.0
Fringe roster player	9.0-11.0
Player who won't stick in the league	0-9.0

Table 2.1: Player productivity and scale based on PER. *Referring from `https://en.wikipedia.org/wiki/Player_efficiency_rating`.*

While PER is a scalable, interpretable and relatively comprehensive metric to evaluate player performance, it still suffers from criticism such that PER is not a reliable measure of a player's defensive acumen, because it largely measures offensive performance but only taking two defensive variables: blocks and steals in the formula.

In this thesis, we use PER as the target variable to build the M5 regression trees using drafted NBA player college datasets, following [4].

Win Shares

Similar to Win Shares in baseball and ice hockey, the win shares in basketball can be divided to two categories: offensive win shares and defensive win shares. Offensive win shares are calculated using points produced and offensive possessions, where the offensive possessions are predicted for each player. *(An offensive possession will end when a) the team scores, b) the team misses and the opponent gets the rebound, c) the team turns over the ball, or d) shooting free throws and either making the last shot or not securing the offensive rebound.)* Using these numbers from a game, the total number of possessions can be estimated for that game. In contrast, defensive win shares are calculated through defensive rating, which is concerned with opponent points and opponent possessions [4].

However, since the win shares takes broad statistics from player, team, and league-wide in the formula, it may be an unfair measurement for a good player who is in a bad team according to the Pythagorean Theory. In our experiments, we also tried using career win shares as a response variable in our tree models. However, it only produced a single regression model and has lower ranking correlation results compared to PER, so we don't present it in this thesis.

Chapter 3

Datasets Description and Exploration

In this chapter, we first describe our datasets and then discuss the distribution of some import attributes with respect to their relationship with target variables. Python 2.7 is used for data colletion, preprocessing and statistical analysis. As for plots, we use the *ggplot2* library in R.

3.1 Data Fields Explanation

3.1.1 Ice Hockey Datasets

Our ice hockey data was obtained from public-domain on-line sources, including `www.nhl.com`, `www.eliteprospects.com`, and `www.draftanalyst.com`. We are also indebted to David Wilson for sharing his NHL performance dataset [33]. The full dataset is posted on the worldwide web*https://github.com/liuyejia/Model_Trees_Full_Dataset*. We consider players drafted into the NHL between 1998 to 2008 (excluding goalies). Following [6], we took as our dependent variable the total number of games g_i played by a player i after 7 years under an NHL contract. The first seven seasons are chosen because NHL teams have at least seven-year rights to players after they are drafted [24]. Our dataset includes also the total time on ice after 7 years. The results for time on ice were very similar to number of games, so we discuss only the results for number of games. The independent variables include demographic factors *(e.g., age)*, performance metrics for the year in which a player was drafted (e.g., goals scored), and the rank assigned to a player by the NHL Central Scouting Service (CSS). Table 3.1 lists all data columns and their meaning. Figure 3.1 shows an excerpt from the dataset.

3.1.2 Basketball Datasets

Our basketball datasets are obtained from `www.basketball-reference.com`, an exhaustive resource of NBA player data, containing both their pre-draft and career information. We

Variable Name	Description
id	nhl.com id for NHL players, otherwise Eliteprospects.com id
DraftAge	Age in Draft Year
Country	Nationality. Canada -> 'CAN', USA -> 'USA', countries in Europe -> 'EURO'
Position	Position in Draft Year. Left Wing -> 'L', Right Wing -> 'R', Center -> 'C', Defencemen -> 'D'
Overall	Overall pick in NHL Entry Draft
CSS_rank	Central scouting service ranking in Draft Year
rs_GP	Games played in regular seasons in Draft Year
rs_G	Goals in regular seasons in Draft Year
rs_A	Assists in regular seasons in Draft Year
rs_P	Points in regular seasons in Draft Year
rs_PIM	Penalty Minutes in regular seasons in Draft Year
rs_PlusMinus	Goal Differential in regular seasons in Draft Year
po_GP	Games played in playoffs in Draft Year
po_G	Goals in playoffs in Draft Year
po_A	Assists in playoffs in Draft Year
po_P	Points in playoffs in Draft Year
po_PIM	Penalty Minutes in playoffs in Draft Year
po_PlusMinus	Goal differential in playoffs in Draft Year
sum_7yr_GP	Total NHL games played in player's first 7 years of NHL career
sum_7yr_TOI	Total NHL Time on Ice in player's first 7 years of NHL career
GP_7yr_greater_than_0	Played a game or not in player's first 7 years of NHL career

Table 3.1: Player Attributes listed in dataset *(excluding weight and height)*.

id	Player Name	Draft Age	Coun try	Hei ght (in)	Wei ght (lbs)	Posi tion	Over all	CSS_ rank	rs _GP	rs_ G	rs_ A	rs_ P	rs_ PIM	rs_ Plus Minus	sum_ 7yr_ GP	sum_ 7yr_ TOI	GP_ 7yr > 0
847-4141	Patrick Kane	19	USA	71	177	R	1	2	65	67	87	154	94	44	515	9927	yes
847-3419	Brad Marchand	18	CAN	69	181	L	71	80	68	29	37	66	83	40	218	3418	yes
27	Yared Hagos	18	EURO	73	218	C	70	24	43	11	26	37	24	1	0	0	no

Figure 3.1: Sample Player Data for their draft year. rs = regular season. We use the same statistics for the playoffs *(not shown)*.

consider players who got drafted into NBA between 1985 and 2011, inclusive. This draft range ensures that a player has enough time(at least 7 years) to accumulate his career performance statistics. Following [4], we choose career PER as our response variable. Our datasets also include career win shares and ws_48. However, when applying M5 regression tree to these two target variables, they only produces a single node with weaker predictive power *(correlation)* than using the career PER, so we don't present them in the thesis.

In our experiment, the datasets are divided into training datasets*(1985-2005 drafts)* and testing datasets*(2006-2011 drafts)* according to the ratio 6/4. Table 3.1 lists all the data columns and their meanings.

Variable	Description
age	Player age in his draft year
height	Player height in his draft year
weight	Player weight in his draft year
position	Player position in his draft year
shoots	Player shoots style, left-handed or right-handed
ah	If a player wined amateur honor in college before he is drafted, then the value is 1, otherwise, 0
g	Games played by the player in his draft year
mp	Minutes played in the player draft year *(total and per game statistics in the player draft year are both collected)*
fg	Field goals gained by the player in his draft year *(total and per game statistics in the draft year are both collected)*
fga	Field goals attempts made by the player in his draft year
3p	3-point field goals obtained by the player in his draft year
3pa	3-point field goal attempts made by the player in his draft year
ft	Free throws made by the player in his draft year *(total and per game statistics in the player draft year are both collected)*
fta	Free throw attempts made by the player in his draft year
orb	Offensive rebounds made by the player in his draft year
trb	Total rebounds made by the player in his draft year *(total and per game statistics are both collected)*
ast	Assists made by the player in his draft year *(total and per game statistics are both collected)*
stl	Steals made by the player in his draft year
blk	Blocks made by the player in his draft year
tov	Turnovers of the player in his draft year
pf	Player personal fouls in his draft year
pts	Points gained by the player in his draft year *(total and per game statistics are both collected)*

Table 3.2: Player Attributes listed in datasets.

3.2 Data Exploration

In this section, we mainly explore the distribution of some important predictors and their relationship with the response variable in our obtained ice hockey and basketball datasets, respectively.

3.2.1 Features Analysis for Ice Hockey Datasets

CSS_rank. The CSS rank each year is given by the full-time professional scouts in NHL Central Scouting Bureau. They rank players based on how well they will translate to the professional game in the National Hockey League. The CSS rank is stratified by player position(Skaters versus Goalies) and player location(North America versus Europe). In [6], the CSS rank played an import role in predicting player career performance. It was converted to Cescin(multiply 1.35 for North American players while 6.27 for European players) for each player. In our experiment, we use the original CSS_rank directly since the position and country are also considered in our model trees. Figure 3.1 shows the non-linear relationship between CSS_rank and sum_7yr_GP.

Country_group and major junior league. The distribution of player sum_7yr_GP grouped by country_group and major junior league OHL, QMJHL, WHL is shown in Table 3.2 and Table 3.3. Notice that players from Canada have higher sum_7yr_GP compared to American and European players, along with a bigger size. For major junior league, the players from OHL perform better than players from other leagues based on their statistics displayed in Table 3.3.

country_group	size	mean	std	min	25%	50%	75%	max
CAN	903.0	66.75	116.21	0.0	0.0	0.0	86.50	524.0
EURO	856.0	50.85	102.74	0.0	0.0	0.0	34.25	475.0
USA	460.0	57.79	105.73	0.0	0.0	0.0	68.0	515.0

Table 3.3: Statistic overview of country_group vs.sum_7yr_GP.

League	size	mean	std	min	25%	50%	75%	max
OHL	352.0	84.12	129.80	0.0	0.0	3.5	132.75	524.0
QMJHL	218.0	55.12	112.24	0.0	0.0	0.0	39.50	471.0
WHL	344.0	70.28	115.27	0.0	0.0	0.0	103.00	494.0
others	1305.0	49.5	99.29	0.0	0.0	0.0	36.00	504.0

Table 3.4: Statistic overview of major league vs.sum_7yr_GP.

3.2.2 Features Analysis for Basketball Datasets

position. Every basketball player has a label to describe what they should do in the court. We call it position, which is usually decided by player physical size and skills. For example, if a player is big and strong, then he is likely to be a *center* or *power forward*. If he is a guard and shoots well, he is potentially a *shooting guard*. Different position contributes differently to the wins. According to the report from Mazique [21], the bigs*(power forwards and centers)* carry the responsibility of scoring and defensing the team, contributing most to the team success. Also, a truly transcendent player is required on the wing*(small forwards and shooting guards)* to elevate the team. Based on the importance of player position in

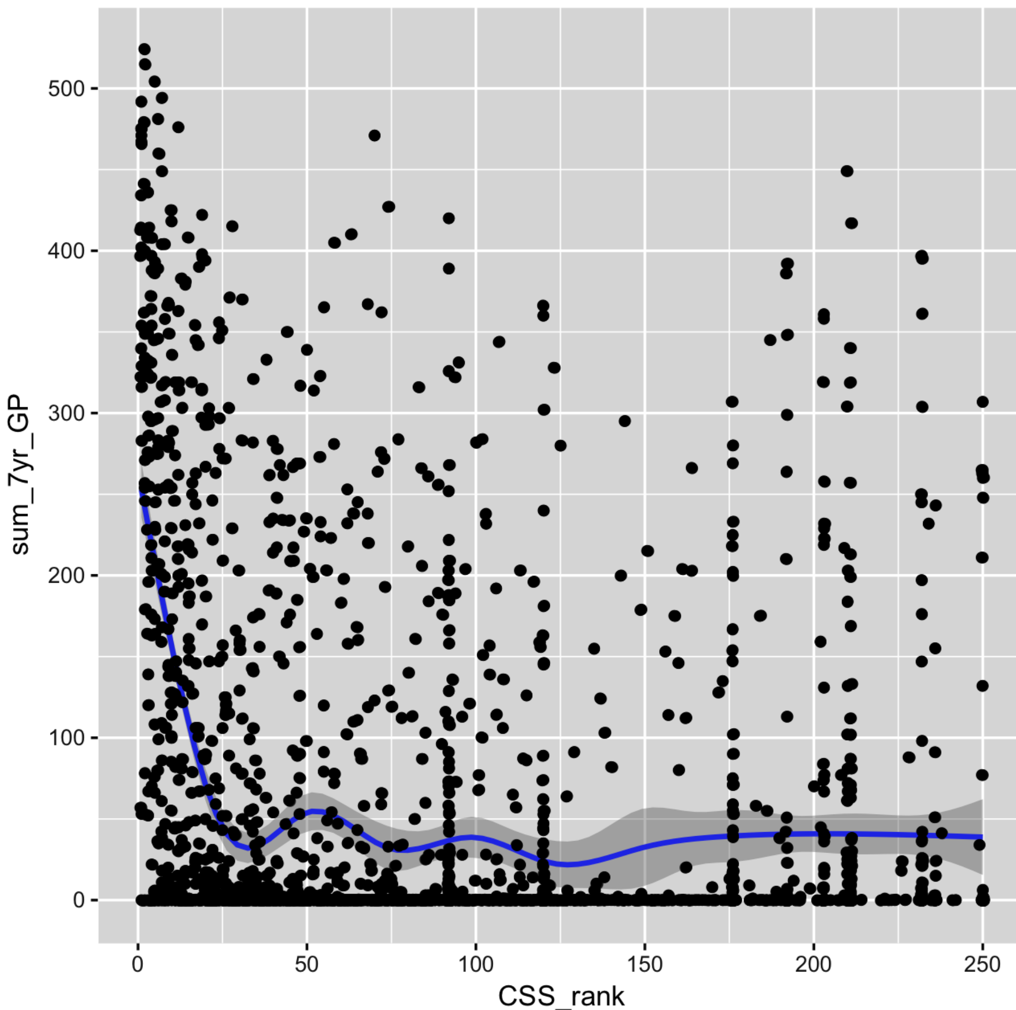

Figure 3.2: Scatter plot of CSS_rank vs.sum_7yr_GP. *Smoothed by generalized additive model.*

previous studies, we analyzes the relationship between the player draft position and career PER in our datasets, shown in Table 3.4. From Table 3.4, we can see *center and power forward* indeed has the highest career PER, then it's *(shooting guard and point guard)*, in accord with most studies for position in the NBA world.

position	size	mean	std	min	25%	50%	75%	max
Center and Power Forward	162	14.93	3.5	7	11.85	13.75	16.33	24.6
Shooting Guard and Point Guard	115	13	3.28	4.4	10.7	12.65	14.75	24.2
Power Forward and Small Forward	108	12.94	3.35	-1.5	11.1	13.35	15.9	20.8
Small Forward and Shooting Guard	68	12.81	3.03	7	10.68	12.65	14.45	25.2
Point Guard	181	12.61	6.7	-6.8	9.7	12.2	15.1	76.1
Power Forward	134	11.95	6.94	-30.2	9.8	11.85	14.78	58.3
Small Forward	142	11.05	4.77	-5.6	8.7	11.05	13.9	31.3
Shooting Guard	142	10.66	4.9	-11.4	8.58	11.45	13.4	22.2
Center	227	9.96	9.78	-48.6	8.6	11.3	13.8	66.8
Guard	10	-14.2	18.97	-57.62	-24.25	-7.87	-1.07	1.61
Forward	12	-14.87	20.73	-57.62	-15.13	-5.48	-1.44	-0.88
Forward/Center	7	-14.24	13.58	-27.62	-27.62	-15.13	-1.63	1.61

Table 3.5: Overview of position and career PER statistical analysis, sorted by the mean career PER value of each position.

Chapter 4

Model Trees Construction and Results

In this chapter, we discuss the construction of model trees and how we apply them to our datastes. we also present and analyze our modelling results and learned groups, with respect to dependent and independent variables. Then, we show how the exceptional players(and underestimated players) and their strongest points discovered by our methods. In our experiment, Python 2.7 and Weka are used to build models and methods. MySQL is used to store our datasets.

4.1 Logistic Model Trees

Logistic Model Tree(LMT) has been created with the purpose of combining advantages of tree induction and linear logistic regression. In each leaf node, explicit class probability estimates can be calculated rather than just a classification label. In the logistic variant, the LogitBoost algorithm is adapted to produce a logistic regression model in every node, and then the tree is split based on C4.5 splitting criterion. LMT has been tested on 36 datasets from UCI repositories [1], which shows its classification accuracy is better than C4.5, CART, Naïve Bayes trees and Lotus [18]. In our thesis, the LMT is used to produce ice hockey prospect groups with a predictive model in each group. In the following subsection, we overview the relevant concepts and algorithms to LMT.

4.1.1 LogitBoost Algorithm

In LMT, every node is a logistic model. To estimate the parameters of logistic model, the LogitBoost algorithm has been applied, which looks for the maximum likelihood estimates of observed data points. The pseudocode for this algorithm is shown in Figure 2.1. The variable y^*_{ij} represents the observed class probabilities for instance x_i. The $p_j(x_i)$ are the estimates of the class probabilities for an instance x given by the model so far.

LogitBoost (J classes)

1. Start with weights $w_{ij} = 1/n,\ i = 1, \ldots, n,\ j = 1, \ldots, J,\ F_j(x) = 0$ and $p_j(x) = 1/J \ \ \forall j$
2. Repeat for $m = 1, \ldots, M$:
   (a) Repeat for $j = 1, \ldots, J$:
      i. Compute working responses and weights in the jth class
      $$z_{ij} = \frac{y^*_{ij} - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))}$$
      $$w_{ij} = p_j(x_i)(1 - p_j(x_i))$$
      ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression of z_{ij} to x_i with weights w_{ij}
   (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x)),\quad F_j(x) \leftarrow F_j(x) + f_{mj}(x)$
   (c) Update $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$
3. Output the classifier $\underset{j}{\operatorname{argmax}}\ F_j(x)$

Figure 4.1: LogitBoost Algorithm [9].

LogitBoost fits a regression model at every boosting step: it computes 'response variable' z_{ij} which encodes the error of the currently fitting model on the training data points (in terms of probability estimates), and then tries to improve the model by adding a function f_{mj} to F_j, fitting to the response by least-squared error. As shown in [9], this process is similar to performing a quasi-Newton step in every iteration. LMT adapted the LogitBoost by adding a constant when the $f_{mj}(x)$ and so the $F_j(x)$ are linear functions of the input variables, since $\sum\limits_{k=1}^{J} f_k(x)$ is zero in this case [17].

4.1.2 Splitting Strategies

LMT uses almost the same splitting strategies applied in C4.5, which involves the concept of information entropy. The attribute with the highest normalized information gain is chosen to make the splitting decision each time. The pseudocode of C4.5 is summarized in the following box [16].

1. Check for the following bases:
   (a) When all the samples in the list belong to the same class, a leaf node for that class is created;
   (b) When none of the attributes provide any information gain, a decision node higher up the tree is created using the expected value of that class;
   (c) When encountering a previously-unseen class, a decision node higher up the tree is created using the expected value of that class.
2. For each attribute a, find the normalized information gain ratio from splitting on *a*.
3. Let a_best be the attribute with the highest normalized information gain.
4. Create a decision node that splits on *a_best*.
5. Recur on the subsets obtained by splitting on *a_best*, and added these node as children of the node.

In the above algorithm, the information gain ratio(IGR) is calculated by information gain(IG) dividing intrinsic value(IV). The IG and IV are defined as follows:

$$IG(Ex,a) = H(Ex) - \sum_{v \in values(a)} \left(\frac{|x \in Ex | value(x,a) = v|}{|Ex|} \cdot H(x \in Ex | value(x,a) = v)\right)$$

$$IV(Ex,a) = - \sum_{v \in values(a)} \left(\frac{|x \in Ex | value(x,a) = v|}{|Ex|} \cdot \log_2\left(\frac{|x \in Ex | value(x,a) = v|}{|Ex|}\right)\right)$$

where Ex is the set of all training examples, $value(x,a)$ denotes the value of a specific example x for an attribute a and $value(a)$ function defines the set of all possible values of the attribute a. In the above formulas, H represents the entropy [25], a measure of unpredictability of data values.

When applying C4.5 algorithm, LMT makes two adjustments to grow more reliable model trees. First, if a node contains less than 15 examples, no splitting would happen. Since the leaves of LMT are complex models, more examples are required for model fitting. Second, a logistic model is only built at a node which contains at least 5 examples, which is the minimum number of examples required by cross-validation in LogitBoost to determine the best number of iterations.

4.1.3 Tree Pruning

LMT employs the pruning method from CART algorithm to avoid overfitting [2]. It uses a combination of training error and penalty term for model complexity to make pruning decisions.

4.2 NHL Predictive Models and Results

In this section, we present and discuss our experiment results. We first describe how we preprocess our datasets. Then, we show the logistic model tree and predicted correlation results, in comparison with actual draft order. Last but not least, we analyze the learned groups with respect to dependent and independent variables, also discuss their interactions.

4.2.1 Data Preprocessing

Some players information is not available online. This issue reflects most in *CSS_rank* and *rs_plusminus*. If a player was not ranked by the Central Scouting Service(CSS), we assign **(1+ the maximum rank for his draft year)** to his CSS rank value. When it comes to the missing rs_plusminus values, we replace them by 0. Another main preprocessing step is to pool all European countries into a single category. Additionally, if a player played for more than one team in his draft year (e.g., a league team and a national team), we add up this counts from different teams.

4.2.2 Model Trees Construction

Model trees are a flexible formalism that can be built for any regression model. An obvious candidate for a regression model would be linear regression; alternatives include a generalized additive model [6], and a Poisson regression model specially built for predicting counts [23]. We introduce a different approach: a logistic regression model to predict whether a player will play any games at all in the NHL ($gi > 0$). The motivation is that many players in the draft never play any NHL games at all (up to 50% depending on the draft year) [31]. This poses an extreme zero-inflation problem for any regression model that aims to predict directly the number of games played. In contrast, for the classification problem of predicting whether a player will play any NHL games, zero-inflation means that the data set is balanced between the classes. This classification problem is interesting in itself; for instance, a player agent would be keen to know what chances their client has to participate in the NHL. The logistic regression probabilities $p_i = P(g_i > 0)$ can be used not only to predict whether a player will play any NHL games, but also to rank players such that the ranking correlates well with the actual number of games played. Our method is therefore summarized as follows.

1. Build a tree whose leaves contain a logistic regression model.
2. The tree assigns each player i to a unique leaf node l_i, with a logistic regression model $m(l_i)$.
3. Use $m(l_i)$ to compute a probability $p_i = P(g_i > 0)$.

Figure 1.1 shows the logistic regression model tree learned for our second cohort by the LogiBoost algorithm. It places CSS rank at the root as the most important attribute.

Players ranked better than 12 form an elite group, of whom almost 82% play at least one NHL games. For players at rank 12 or below, the tree considers next their regular season points total. Players with rank and total points below 12 form an unpromising group: only 16% of them play an NHL game. Players with rank below 12 but whose points total is 12 or higher, are divided by the tree into three groups according to whether their regular season plus-minus score is positive, negative, or 0. (A three-way split is represented by two binary splits). If the plus-minus score is negative, the prospects of playing an NHL game are fairly low at about 37%. For a neutral plus-minus score, this increases to 61%. For players with a positive plus-minus score, the tree uses the number of playoff assists as the next most important attribute. Players with a positive plus-minus score and more than 10 playoff assists form a small but strong group that is 92% likely to play at least one NHL game.

4.2.3 Modelling Results

Following [6], we evaluated the predictive accuracy of the LMT model using the Spearman Rank Correlation(SRC) between two player rankings: *i*) the performance ranking based on the actual number of NHL games that a player played, and *ii*) the ranking of players based on the probability *pi* of playing at least one game(Tree Model SRC). We also compared it with *iii*) the ranking of players based on the order in which they were drafted (Draft Order SRC). The draft order can be viewed as the ranking that reflects the judgment of NHL teams. We provide the formula for the Spearman correlation in the Appendix A. Table 4.1 shows the Spearman correlation for different rankings.

Training Data NHL Draft Years	Out of Sample Draft Years	Draft Order SRC	Tree Model Classification Accuracy	Tree Model SRC
1998, 1999, 2000	2001	0.43	82.27%	0.83
1998, 1999, 2000	2002	0.30	85.79%	0.85
2004, 2005, 2006	2007	0.46	81.23%	0.84
2004, 2005, 2006	2008	0.51	63.56%	0.71

Table 4.1: Predictive Performance (our model, over all draft ranking) using Spearman Rank Correlation. Bold indicates the best values.

Other Approaches. We also tried designs based on a linear regression model tree, using the M5P algorithm implemented in the Weka program. The result is a decision stump that splits on CSS rank only, which had substantially worse predictive performance (i.e., Spearman correlation of only 0.4 for the $2004 - 2006$ cohort). For the generalized additive model (gam), the reported correlations were $2001 : 0.53, 2002 : 0.54, 2007 : 0.69, 2008 : 0.71$ [6]. Our correlation is not directly comparable to the gam model because of differences in data preparation: the gam model was applied only to drafted players who played at least one NHL game, and the CSS rank was replaced by the Cescin conversion factors: for North American players, multiply CSS rank by 1.35, and for European players, by 6.27 [10]. The

Cescin conversion factors represent an interaction between the player's country and the player's CSS rank. A model tree offers another approach to representing such interactions: by splitting on the player location node, the tree can build a different model for each location. Whether the data warrant building different models for different locations is a data-driven decision made by the tree building algorithm. The same point applies to other sources of variability, for example the draft year or the junior league. Including the junior league as a feature has the potential to lead to insights about the differences between leagues, but would make the tree more difficult to interpret; we leave this topic for future work. In the next section we examine the interaction effects captured by the model tree in the different models learned in each leaf.

4.2.4 Groups and Variables Interaction

In this section, we examine the learned group regression models, first in terms of the dependent success variable, then in terms of the player features.

Learned Groups and Dependent Variable. Figure 4.2 shows boxplots for the distribution of our dependent variable g_i. The strongest groups are, in order, 1, 6, and 4. The other groups show weaker performance on the whole, although in each group some players reach high numbers of games. Most players in Group 2&3&4&5 have GP equals to zero while Group 1&6 represent the strongest cohort in our prediction, where over 80% players played at least 1 game in NHL. The tree identifies that among the players who do not have a very high CSS rank (worse than 12), the combination of regular season $Points >= 12$, $PlusMinus > 0$, and $play-offAssists > 10$ is a strong indicator of playing a substantive number of NHL games (median $g_i = 128$).

Learned Groups and Independent Variables. Figure 4.3 shows the average statistics by group and for all players. The CSS rank for Group 1 is by far the highest. The data validate the high ranking in that 82% players in this group went on to play an NHL game. Group 6 in fact attains an even higher proportion of 92%. The average statistics of this group are even more impressive than those of group 1 (e.g., 67 regular season points in group 6 vs. 47 for group 1). But the average CSS rank is the lowest of all groups. So this group may represent a small group of players ($n = 13$) overlooked by the scouts but identified by the tree. Other than Group 6, the group with the lowest CSS rank on average is Group 2. The data validate the low ranking in that only 16% of players in this group went on to play an NHL game. The group averages are also low (e.g., 6 regular season points is much lower than other groups).

Learned Group Weights and Variable Interactions. Figure 4.4 illustrates logistic regression weights by group. A positive weight implies that an increase in the covariate value predicts a large increase in the probability of playing more than one game, compared to the probability of playing zero games. Conversely, a negative weight implies that an increase in the covariate value decreases the predicted probability of playing more than one game. Bold

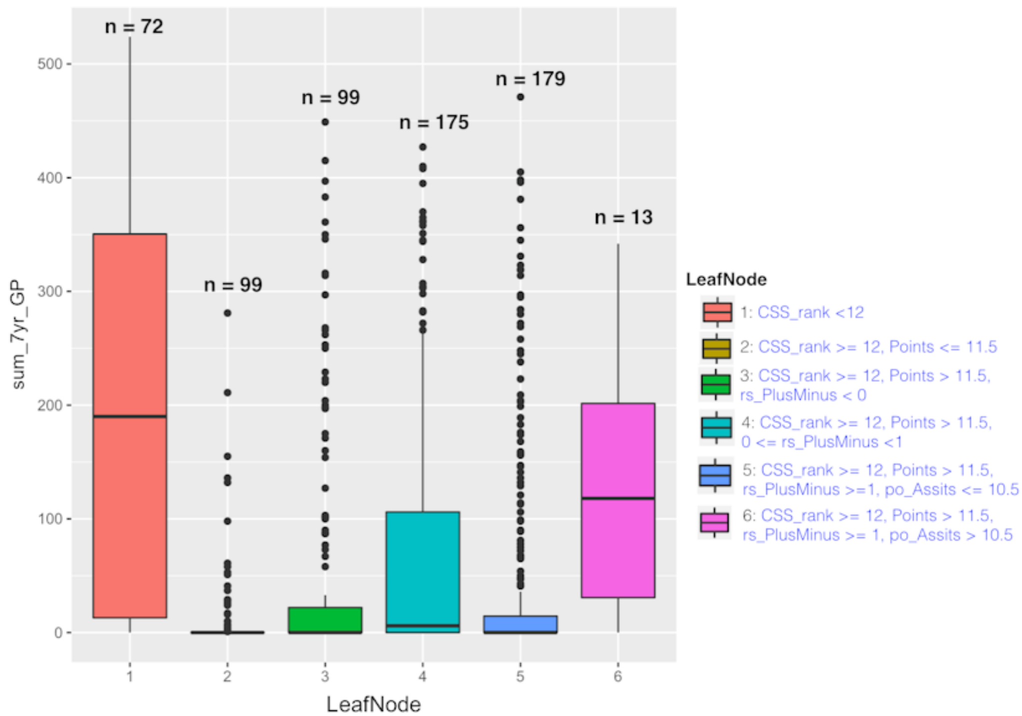

Figure 4.2: Boxplots for the dependent variable g_i , the total number of NHL games played after 7 years under an NHL contract. Each boxplot shows the distribution for one of the groups learned by the logistic regression model tree. The group size is denoted n.

Group	Mean Points											
	rs_P	rs_A	CSS_ rank	po_A	Weight	Height	rs_ Plus Minus	rs_GP	rs_ PIM	Draft Age	po_ GP	po_P
1	47	27	7	4	204	74	6	55	63	18	8	7
2	6	4	94	1	206	74	-2	38	56	18	8	1
3	40	23	76	2	201	73	-9	63	78	18	7	4
4	44	25	86	4	198	73	0	47	59	19	10	8
5	43	26	71	3	199	73	12	62	73	19	9	5
6	67	44	107	14	193	71	23	65	83	19	19	19
all	*39*	*23*	*101*	*2*	*201*	*73*	*3*	*55*	*67*	*18*	*6*	*4*

Figure 4.3: Statistics for the average players in each group and all players.

numbers show the groups for which an attribute is most relevant. The table exhibits many interesting interactions among the independent variables; we discuss only a few. Notice that if the tree splits on an attribute, the attribute is assigned a high-magnitude regression weight by the logistic regression model for the relevant group. Therefore our discussion focuses on the tree attributes.

Metrics / Group	CSS_ rank	Draft Age	Height	Weight	Country _group	Position	Games _played	Goals	Assists	Points	Penalty_ in_ Minutes	Plus Minus
1	**-17.9**	-3.91	-2.69	2.35	E: -0.77 C: 1.23 U:-0.49	D: -0.54 L: 2.09 R: -0.68	rs: -2.43 po: 4.15	rs: -0.03 po: **-9.8**	rs: 1.97 po: 8.89	rs: 1.73 po: 0.3	rs: **7.98** po: **-7.6**	rs: - 0.45
2	-1.12	-1.1	-4.8	**6.7**	E: -0.13 C: 0.28 U:-0.22	D: -1.1 L: -0.45 R: 1.89	rs: 5.9 po: - 14.1	rs: -2.17 po: -2.9	rs: **11.8** po: **21.6**	rs: **14.2** po:**11.1**	rs: -2.72 po: 5.2	rs: 1.57
3	-2.6	**6.95**	-7.4	**6.7**	E: -2.4 C: 1.04 U: 2.34	D: 0.39 L: 0.68 R: -0.4	rs: 3.21 po: - 1.05	rs:-0.52 po: -0.6	rs: 1.36 po: -0.39	rs: 0.54 po:-2.6	rs: -1.88 po: 2.7	rs: 13.16
4	-2.4	5.2	-4.2	-0.52	E: 1.08 C: -0.40 U: -0.6	D: -0.03 L: -0.24 R: 0.14	rs: 3.58 po: -4.5	rs: -2.16 po: 1.58	rs: -0.12 po: 1.71	rs: -1.4 po: 1.6	rs: -2.72 po: 3.45	rs: 0
5	-0.65	-3.89	**0.01**	4.68	E: -1.26 C: 0.74 U: 0.47	D: 0.91 L: -0.64 R: 0.05	rs: 2.24 po: - 0.25	rs: 3.59 po: -1.7	rs: -0.23 po: 0.33	rs: 2.19 po:-0.8	rs: -4.05 po: 6.86	rs: - 0.73
6	-8.89	6.64	-14.91	0.34	E: -28.1 **C: 5.9** U: 7.2	D: 3.32 L: 0.74 R: -28.12	**rs: 16.7** po: 2.74	rs: **21.6** po: -9.7	rs: -0.34 po: -0.43	rs: -0.5 po:-0.4	rs: 1.3 po: -1.6	rs: **21.9**

Figure 4.4: Group 200(4 + 5 + 6 + 7 + 8) Weights Illustration. E = Europe, C = Canada, U = USA, rs = Regular Season, po = Playoff. Largest-magnitude weights are in bold. Underlined weights are discussed in the text.

At the tree root, *CSS rank* receives a large negative weight of -17.9 for identifying the most successful players in Group 1, where all CSS ranks are better than 12. Figure 4.5a shows that the proportion of above-zero to zero-game players decreases quickly in Group 1 with worse CSS rank. However, the decrease is not monotonic. Figure 4.5b is a scatterplot of the original data for Group 1. We see a strong linear correlation ($p = -0.39$), and also a large variance within each rank. The proportion aggregates the individual data points at a given rank, thereby eliminating the variance. This makes the proportion a smoother dependent variable than the individual counts for a regression model.

Group 5 has the smallest logistic regression coefficient of -0.65. Group 5 consists of players whose CSS ranks are worse than 12, regular season points above 12, and plus-minus above 1. Figure 4.6a plots CSS rank vs. above-zero proportion for Group 5. As the proportion plot shows, the low weight is due to the fact that the proportion trends downward only at ranks worse than 200. The scatterplot in Figure 4.6b shows a similarly weak linear correlation of -0.12.

Regular season points are the most important predictor for Group 2, which comprises players with CSS rank worse than 12, and regular season points below 12. In the proportion plot Figure 4.7, we see a strong relationship between points and the chance of playing more than 0 games (logistic regression weight 14.2). In contrast in Group 4 (overall weight -1.4), there is essentially no relationship up to 65 points; for players with points between 65 and 85 in fact the chance of playing more than zero games slightly decreases with increasing points.

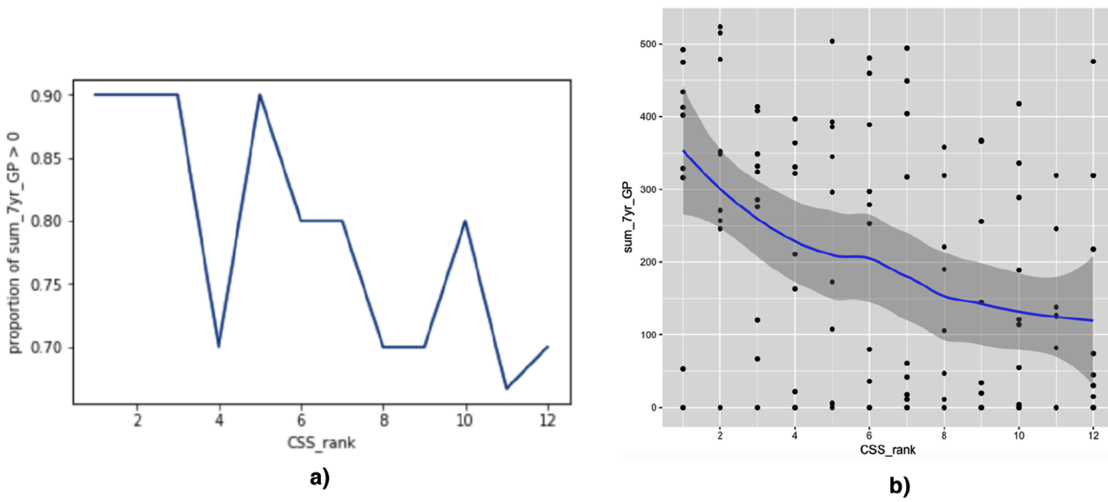

Figure 4.5: Proportion and scatter plots for CSS_rank vs. sum_7yr_GP in Group 1.

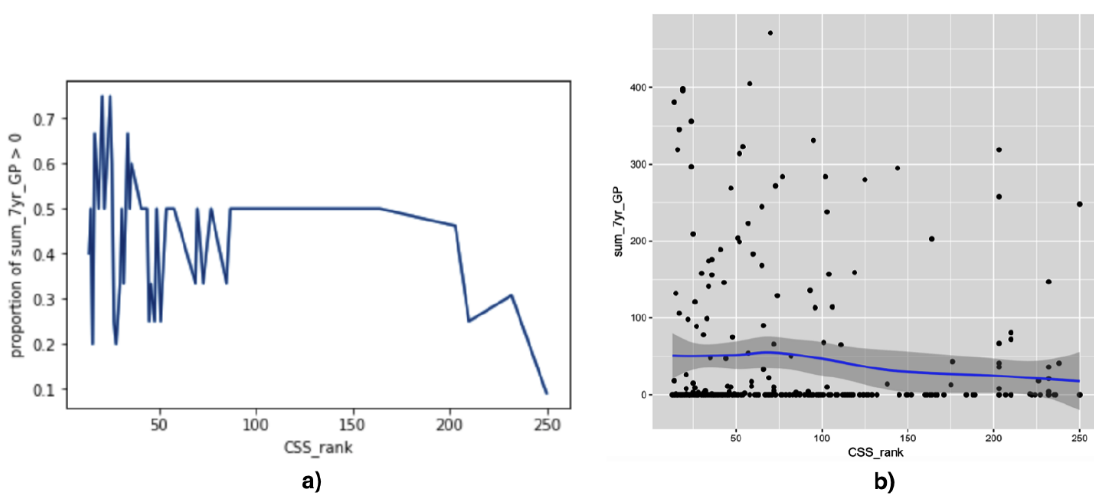

Figure 4.6: Proportion and scatter plots for CSS_rank vs.sum_7yr_GP in Group 5.

In Group 3, players are ranked at level 12 or worse, have collected at least 12 regular season points, and show a negative plus-minus score. The most important feature for Group 3 is the regular season plus-minus score (logistic regression weight 13.16), which is negative for all players in this group. In this group, the chances of playing an NHL game increase with plus-minus, but not monotonically, as Figure 4.8 shows.

For *regular season goals*, Group 5 assigns a high logistic regression weight of 3.59. However, Group 2 assigns a surprisingly negative weight of -2.17. Group 5 comprises players at CSS rank worse than 12, regular season points 12 or higher, and positive plus-minus greater nhl.png

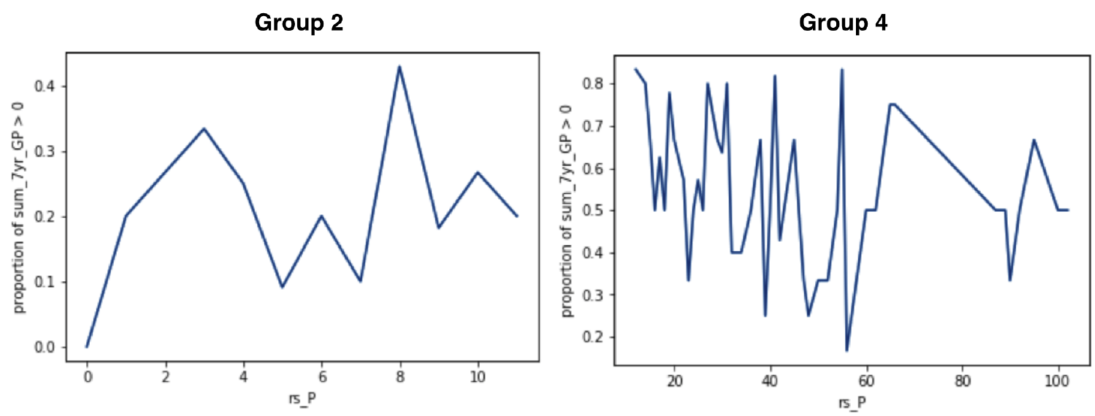

Figure 4.7: Proportion_of_Sum_7yr_GP_greater_than_0 vs. rs_P in Group 2&4.

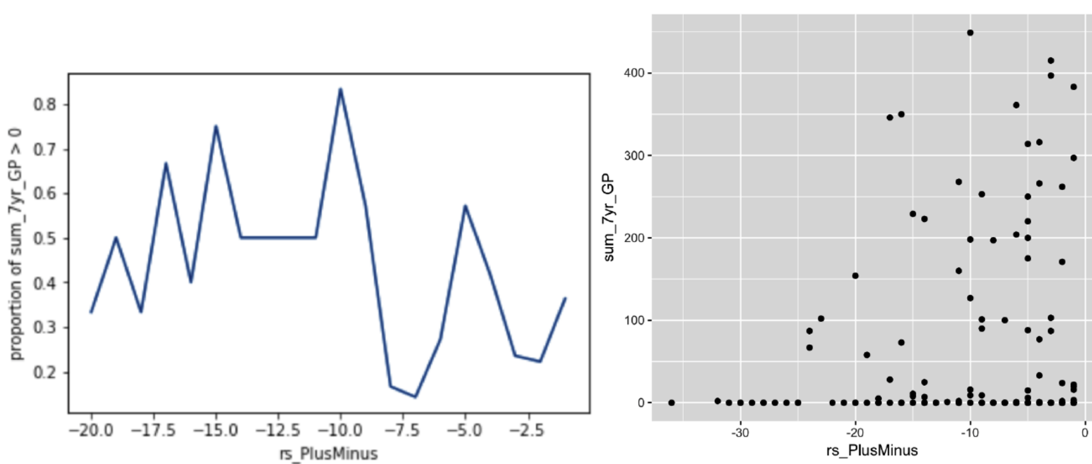

Figure 4.8: Proportion and scatter plots for rs_PlusMinus vs.sum_7yr_GP in group 3.

than 1. About 64.8% in this group are offensive players (see Figure 4.9). The positive weight therefore indicates that successful forwards score many goals, as we would expect.

Group 2 contains mainly defensemen (61.6%; see Figure 4.9). The typical strong defenseman scores 0 or 1 goals in this group. Players with more goals tend to be forwards, who are weaker in this group. In sum, the tree assigns weights to goals that are appropriate for different positions, using statistics that correlate with position (e.g., plus-minus), rather than the position directly.

4.3 NHL Case Studies: Exceptional Players

Teams make drafting decisions not based on player statistics alone, but drawing on all relevant source of information, and with extensive input from scouts and other experts.

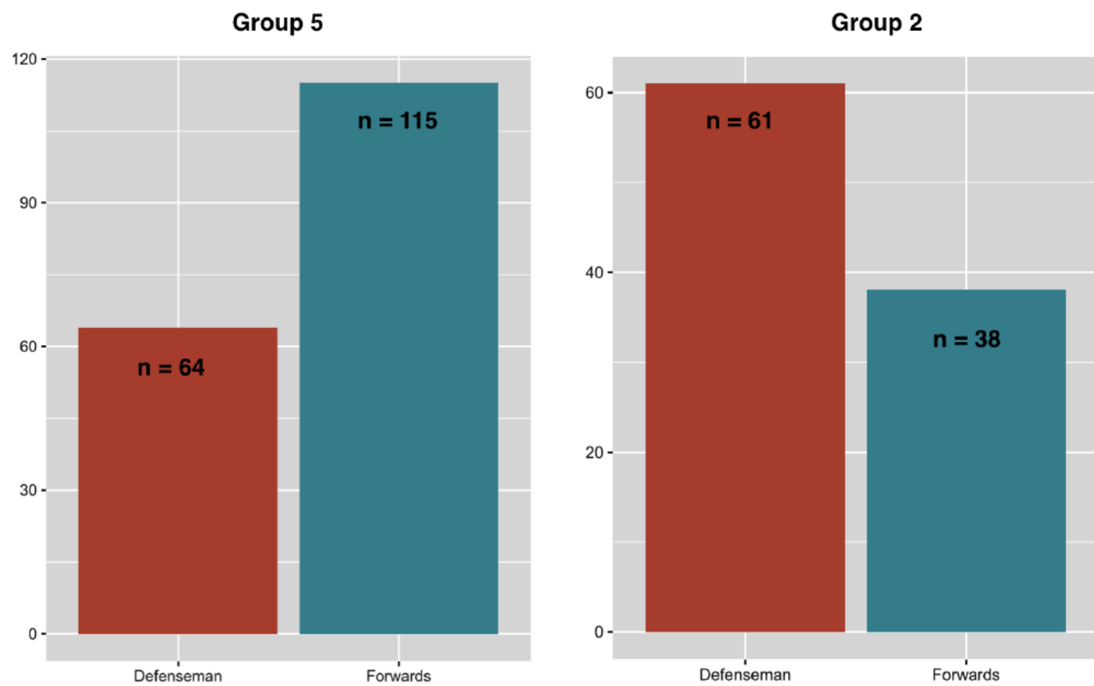

Figure 4.9: Distribution of Defenseman vs. Forwards in Group 5&2. The size is denoted as n.

As Cameron Lawrence from the Florida Panthers put it, 'the numbers are often just the start of the discussion'[13]. In this section we discuss how the model tree can be applied to support the discussion of individual players by highlighting their special strengths. The idea is that the learned weights can be used to identify which features of a highly-ranked player differentiate him the most from others in his group.

4.3.1 Explaining the Rankings: identify weak points and strong points

Our method is as follows. For each group, we find the average feature vector of the players in the group, which we denote by $\overline{x_{g1}}, \overline{x_{g2}}, ..., \overline{x_{gm}}$ (see Table 4). We denote the features of player i as $x_{i1}, x_{i2}, ..., x_{im}$. Then given a weight vector (w_1, w_m) for the logistic regression model of group g, the log-odds difference between player i and a random player in the group is given by

$$\sum_{j=1}^{m} w_j(x_{ij} - \overline{x_{gi}})$$

We can interpret this sum as a measure of how high the model ranks player i compared to other players in his group. This suggests defining as the player's strongest features the x_{ij} that maximize $w_j(x_{ij} - \overline{x_{gi}})$, and as his weakest features those that minimize $w_j(x_{ij} - \overline{x_{gi}})$. This approach highlights features that are i) relevant to predicting future success, as measured by the magnitude of w_j, and ii) different from the average value in the player's group of comparables, as measured by the magnitude of $x_{ij} - \overline{x_{gi}}$.

4.3.2 Case Studies

Table 6 shows, for each group, the three strongest points for the most highly ranked players in the group. We see that the ranking for individual players is based on different features, even within the same group. The table also illustrates how the model allows us to identify a group of comparables for a given player. We discuss a few selected players and their strong points. The most interesting cases are often those where are ranking differs from the scouts'CSS rank. We therefore discuss the groups with lower rank first.

Among the players who were not ranked by CSS at all, our model ranks *Kyle Cumiskey* at the top. Cumiskey was drafted in place 222, played 132 NHL games in his first 7 years, represented Canada in the World Championship, and won a Stanley Cup in 2015 with the Blackhawks. His strongest points were being Canadian, and the number of games played (e.g., 27 playoff games vs. 19 group average).

In the lowest CSS-rank group 6 (average 107), our top-ranked player *Brad Marchand* received CSS rank 80, even below his Boston Bruin teammate's Lucic's. Given his Stanley Cup win and success representing Canada, arguably our model was correct to identify him as a strong NHL prospect. The model highlights his superior play-off performance, both in terms of games played and points scored. Group 2 (CSS average 94) is a much weaker group. *Matt Pelech* is ranked at the top by our model because of his unusual weight, which in this group is unusually predictive of NHL participation. In group 4 (CSS average 86), *Sami Lepisto* was top-ranked, in part because he did not suffer many penalties although he played a high number of games. In group 3 (CSS average 76), *Brandon McMillan* is ranked relatively high by our model compared to the CSS. This is because in this group, left-wingers and shorter players are more likely to play in the NHL. In our ranking, *Milan Lucic* tops Group 5 (CSS average 71). At 58, his CSS rank is above average in this group, but much below the highest CSS rank player (Legein at 13). The main factors for the tree model are his high weight and number of play-off games played. Given his future success (Stanley Cup, NHL Young Stars Game), arguably our model correctly identified him as a star in an otherwise weaker group. The top players in Group 1 like *Sidney Crosby* and *Patrick Kane* are obvious stars, who have outstanding statistics even relative to other players in this strong group.

4.4 M5 Regression Trees

M5 model trees(M5P) are designed for tasks to predict a numeric value associated with a case rather than just the class which the case belongs to [14]. In the leaves, a multivariate linear regression model is built instead of just a numeric value. Compared to standard

Group	Top Players	Strongest Points ($\bar{x}$ = group mean)		
1	Sidney Crosby	rs_P	rs_A	CSS_rank
		188 ($\bar{x} = 47$)	110 ($\bar{x} = 27$)	1 ($\bar{x} = 7$)
	Patrick Kane	rs_P	rs_A	CSS_rank
		154 ($\bar{x} = 47$)	87 ($\bar{x} = 27$)	2 ($\bar{x} = 7$)
	Sam Gagner	rs_P	po_A	rs_A
		118 ($\bar{x} = 47$)	22 ($\bar{x} = 4$)	83 ($\bar{x} = 27$)
2	Matt Pelech	Weight	CSS_rank	rs_A
		230 ($\bar{x} = 206$)	41 ($\bar{x} = 94$)	4 ($\bar{x} = 4$)
	Adam Pineault	CSS_rank	rs_P	Height
		25 ($\bar{x} = 94$)	8 ($\bar{x} = 6$)	73 ($\bar{x} = 74$)
	Roman Wick	rs_P	CSS_rank	rs_PlusMinus
		10 ($\bar{x} = 6$)	36 ($\bar{x} = 94$)	0 ($\bar{x} = -2$)
3	A.J.Jenks	CSS_rank	Weight	Country
		20 ($\bar{x} = 76$)	205 ($\bar{x} = 201$)	USA
	Bill Sweatt	CSS_rank	Position	rs_PlusMinus
		27 ($\bar{x} = 76$)	L	-1 ($\bar{x} = -2$)
	Brandon McMillan	CSS_rank	Position	Height
		44 ($\bar{x} = 76$)	L	71 ($\bar{x} = 73$)
4	Sami Lepisto	CSS_rank	rs_GP	rs_PIM
		25 ($\bar{x} = 86$)	61 ($\bar{x} = 47$)	30 ($\bar{x} = 59$)
	Linus Omark	CSS_rank	Height	DraftAge
		55 ($\bar{x} = 86$)	70 ($\bar{x} = 73$)	20 ($\bar{x} = 19$)
	Oscar Moller	CSS_rank	Height	rs_GP
		20 ($\bar{x} = 86$)	70 ($\bar{x} = 73$)	68 ($\bar{x} = 47$)
5	Milan Lucic	Weight	po_GP	CSS_rank
		236 ($\bar{x} = 199$)	23 ($\bar{x} = 9$)	58 ($\bar{x} = 71$)
	Michael Del Zotto	Position	Country	po_GP
		D	CAN	15 ($\bar{x} = 9$)
	Steven Delisle	Weight	Country	po_GP
		234 ($\bar{x} = 199$)	CAN	19 ($\bar{x} = 9$)
6	Brad Marchand	Country	po_GP	po_P
		CAN	25 ($\bar{x} = 19$)	23 ($\bar{x} = 19$)
	Mathieu Carle	Country	CSS_rank	rs_GP
		CAN	53 ($\bar{x} = 107$)	67 ($\bar{x} = 65$)
	Kyle Cumiskey	Country	po_GP	rs_GP
		CAN	27 ($\bar{x} = 19$)	72 ($\bar{x} = 65$)

Figure 4.10: Strongest Statistics for the top players in each group. Underlined players are discussed in the text.

Classification and Regression Tree(CART), M5P are generally smaller in tree size and more accurate, meanwhile, they can also deal with high dimensionality attributes. In our thesis, the M5P are used to predict the player efficiency rating(PER) for drafted players in NBA based on their college statistics. In the subsections, the construction of M5P is reviewed.

4.4.1 Initial Tree Construction

The growing and splitting of M5P is based on the standard deviation of the target variables in training cases. Supposing we have a set of training examples T, and T_i represents the ith subset of T. A test which determines the subset of cases related to each outcome is carried out in every possible T_i. After examining all the possible test cases, the one with maximum error reduction is chosen. The expected error reduction is defined as:

$$\Delta error = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

where $sd(T)$ is the standard deviation of target values of all training cases and $sd(T_i)$ is the standard deviation of target values of cases in T_i.

4.4.2 Linear Models Development

A multivariate linear model is built at every tree node with standard regression methods. In the construction process, the accuracy of linear models and subtree is compared to make decisions. After the model is obtained, M5P uses a greedy search to remove variables that contribute little to the model to simplify linear models.

4.4.3 Tree Pruning

Starting from the bottom, every non-leaf node is examined. M5P decides either the simplified linear model or the model subtree as the final model for this node, based on the estimated error, given by average residues on these cases multiplying $(n + v)/(n - v)$, where n is the number of training cases and v is the number of parameters in the model.

4.4.4 Smoothing

Smoothing is used in M5P to improve the prediction accuracy [22]. The predicted value of a case given by the model at the leaf is adjusted to reflect the predicted values at nodes along the path from the root to the leaf. The formula of adjustment is defined as follows:

$$PV(S) = \frac{n_i \times PV(S_i) + k \times M(S)}{n_i + k}$$

where S_i is the branch of subtree S, n_i is the number of training cases at S_i, $PV(S_i)$ function denotes the predicted value at S_i, and $M(S)$ is the value given by the model at S and k is a smoothing constant*(default* 15*)*.

4.5 NBA Predictive Models and Results

In this section, we present and discuss our experiment results. First, we describe how we preprocess our datasets. Then, we show the constructed M5 regression tree and predicted correlation results, in comparison with actual draft order and our baseline method*(ordinary* *linear regression)*. Lastly, we analyze the learned groups in terms of the relationship between weights/attributes and career PER.

4.5.1 Data Preprocessing

In our datasets, some players college performance statistics are not available, only their career statistics exists. Since it's unreasonable to predict from nothing, we excluded players belonging to these cases. There are also some players whose career statistics are missing but having college statistics. We replace their career statistics**(PER)** by $min(x) - std(x)$ of their draft year, since we think he may not be good enough to play at all in NBA. For the players who miss both values, we discard them. In Table 4.2, we summarize the count of these players in our datasets.

College stats	NBA stats	count
1	0	15
0	1	173
0	0	35
1	1	1405

Table 4.2: Summary of statistics availability.*1 denotes stats are available, otherwise, it is 0.*

4.5.2 Model Trees Construction

Different from NHL, most drafted basketball players would play at least one game in NBA(over %80 in our datasets). Meanwhile, only a small number of players have career PER above league average. Thus, it's not easy to find a target variable with proper threshold to classify players as what we did for ice hockey candidates. In this situation, we intuitively turn to regression approaches. Our method of constructing M5 regression tree in basketball datasets is enlightened by the logistic model tree structure. It links predictors with continuous response variable directly.

Our method is summarized as follows:

1. Build a tree whose leaves contain a linear regression model.
2. The tree assigns each player i to a unique leaf node I_i, with a linear regression model $m(I_i)$.
3. Use $m(I_i)$ to compute predicted career PER.

Figure 4.2 shows the M5 regression tree for all our datasets. The attribute *position* is placed at the root as the most import attribute, corresponding to previous studies which clustering players by their position [19]. Players who are from *Position_Union_1* form a better group compared to the rest ones, with average PER about 13. For players who are not from Position_Union_1, the tree takes the *age* as the next splitting attribute. Players who are older than 24 years old and not from Position_Union_1, they belong to a less promising group with PER around 10. Then, the tree choose *position* as another splitting point again, reflecting its significance again. For players who not belong to Position_Union_1 but belong to *Position_Union_2*, with age smaller than or equal to 24, they form an average level group, with PER value around 7. Lastly, the tree chooses *blk(blocks)* as a splitting feature, however, since the size of Group 1 and Group 2 is relatively small(8 and 18), we treat them as one single group in our following analysis mostly.

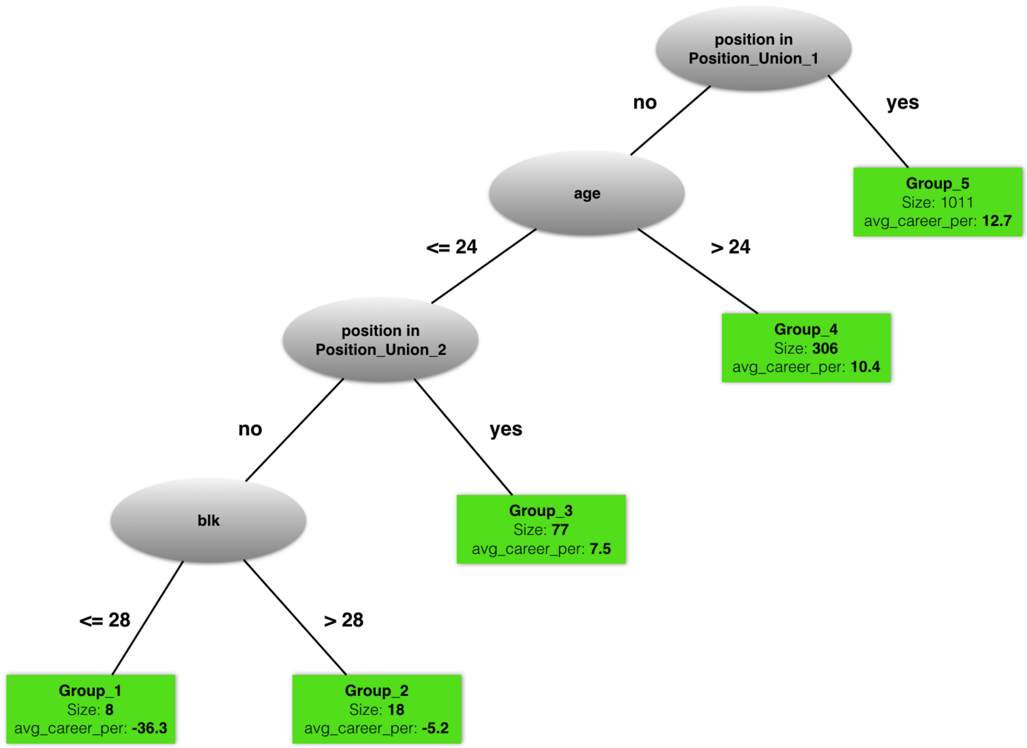

Figure 4.11: M5 Regression Model Trees for all the drafted players in 1985-2011 drafts. *The values of Position_Union_1 and Position_Union_2 are listed in Appendix B.*

Figure 4.3 visualizes the distribution our response variable career PER among each leaf node. Although the size of Group 5 is the largest, the variance between players PER is smaller than the ones in other groups. In order, the strongest groups are 5, 4 and 3, in order.

4.5.3 Modelling Results

To evaluate the predictive results, we use both Pearson Correlation and Spearman Ranking Correlation to compare the predictive power of actual draft order, a baseline *(ordinary linear regression)* and our tree models, displayed in Table 4.2. The result shows our model trees outperform the actual draft order and ordinary linear regression.

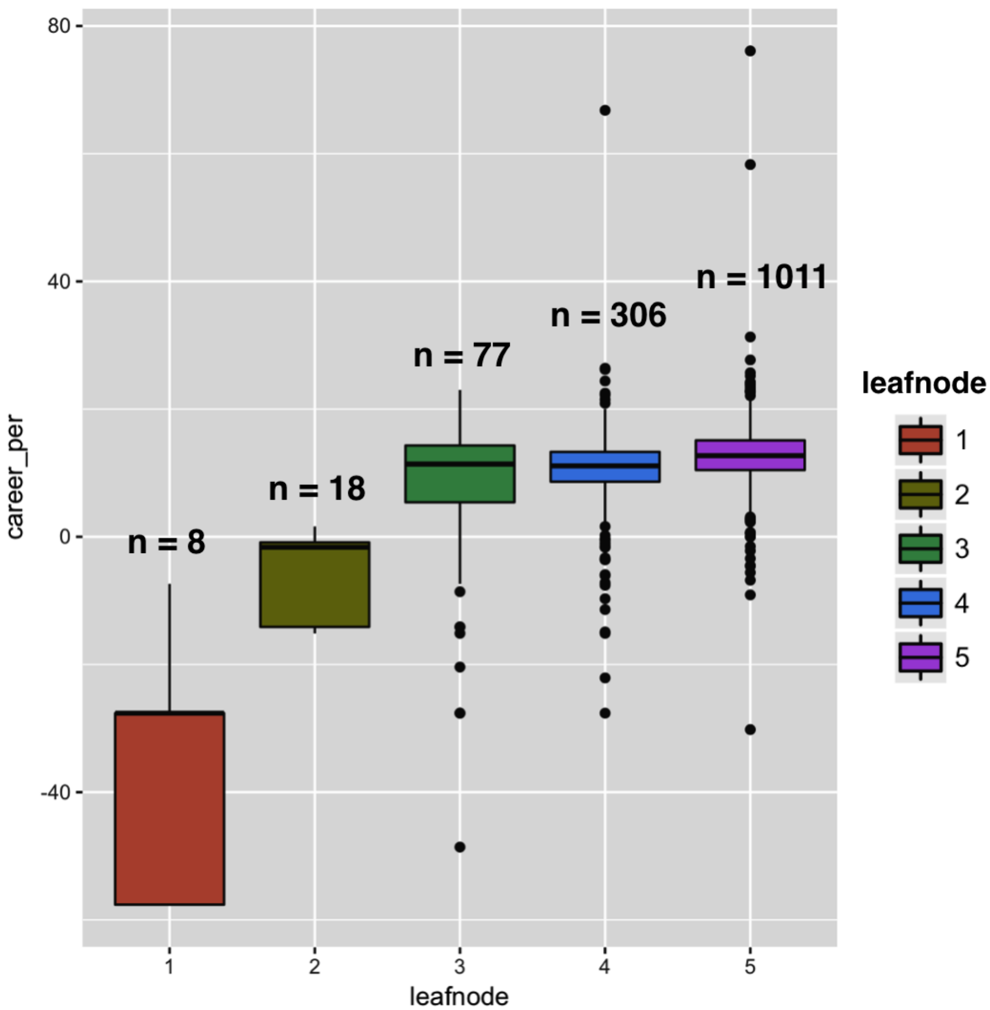

Figure 4.12: Box plots for career PER vs. leaf node. The group size is denoted as n.

4.5.4 Group Models

Table 4.3 illustrates the weights of each learned group. The most relevant attributes which have the largest magnitude are bold and underlined. A positive weight means an increase in the variate value predicts an increase in the predicted career PER, otherwise, it brings decrease in the predicted career PER. It's noteworthy that if tree splits on an attribute, the attribute is assigned a high-magnitude regression weight by the M5 regression regression model for the relevant group, similar to LMT.

For Group 1&2, *blk(blocks)* receives the largest positive weight, in contrast to the one in Group 3. This verifies the splitting node on *blk(blocks)* for Group 1 and Group 2 in the tree. In Group 3, the *trb_per(total rebounds per game)* has the largest positive weight, in contrast with the negative weight in Group 5(the strongest group). *Pts_per(points per game)*

Evaluation / Method	Pearson Correlation	Spearman Rank Correlation	RMSE
Draft Order	0.42	0.39	NaN
Linear Regression	0.45	0.40	7.14
Our Model Trees	**0.55**	**0.43**	**6.16**

Table 4.3: Comparison of predictive performance between draft order, linear regression and our tree models. *Bold indicates the best values*

plays an important role in Group 4, while it impacts little in Group 1&2. In Group 5(the strongest group), *ft(total number of free throws)* is the most import attributes. This results are in accord with the empirical experience in NBA: 'free throws can normally be shot at a high percentage by good players'(`https://en.wikipedia.org/wiki/Free_throw`). *Age* receives the largest negative weight in Group 4, in comparison with Group 1&2 and Group 3, also agreeing with the splitting node *age* in model trees. The weight of *fta(free throw attempts)* is negative among all groups, especially in Group 4 and Group 5, in accord with the real basketball world.

Metrics / Group	age	position	g	mp	ft	fta	trb	ast	blk	pts	ah
1	-0.04	10.95 0.05 0.07	22.78	all: 0 per: 0	all: 2.25 per: -0.14	<u>-1.89</u>	all: 0.21 per: 25.92	all: 0 per: 0.11	**73.31**	all: 0 per: 0.31	0.04
2	-0.04	10.95 0.05 0.07	22.78	all: 0 per: 0	all: 2.25 per: -0.14	<u>-1.89</u>	all: 0.21 per: 25.92	all: 0 per: 0.11	**51.55**	all: 0 per: 0.31	0.04
3	-0.04	6.95 0.05 0.07	10.22	all: 0 per: 0	all: 2.25 per: -0.14	<u>-1.89</u>	all: 0.21 per: **11.55**	all: 0 per: 0.11	1.53	all: 0 per: 0.31	0.04
4	<u>-34.35</u>	1.92 0.05 0.07	12.89	all: 0 per: 5.08	all: 2.25 per: -0.14	-18.63	all: 0.21 per:0	all: 0 per: 0.11	9.51	all: -11.24 per: **17.91**	0.04
5	-2.39	0.36 0.83 0.53 1.34	-6.54	all: 4.27 per: -4.91	all: **10.57** per: -5.33	<u>-10.69</u>	all: 10.05 per: -4.95	all: 5.66 per: 0.04	2.41	all: 2.92 per: 4.01	1.03

Table 4.4: Weights Illustration. Largest weights are in bold. Smallest weights are underlined.

To evaluate the accuracy of model trees in terms of assigning proper weights among each leaf model, we compute the Pearson correlation and p-value for the most relevant attributes selected by the tree, shown in Table 4.4. It illustrates our predictive models are mostly correct in weights assignment, except the *trb_per(rebounds per game)*, which is supposed to have a positive weight in Group 5.

Comparing Group	Comparing Metric	Weights	Pearson Correlation	p-value
Group (1+2) vs. Group 3	blk	73.31, 51.55 vs. 1.53	0.34 vs. 0.09	0.08 vs. 0.44
Group 3 vs. Group 5	trb_per	25.92 vs. -4.95	0.04 vs. 0.10	0.71 vs. 0.001
Group 4 vs. Group (1+2)	pts_per	17.91 vs. 0.31(2)	0.19 vs. 0.3	0.0005 vs. 0.13
Group 5 vs. Group (1+2+3+4)	ft_all	10.57 vs. 2.25(4)	0.12 vs. 0.09	0.0001 vs. 0.04
Group (1+2) vs. Group 3 vs. Group 5	fta	-1.89(2) vs. -1.89 vs. -10.69	0.30 vs. -0.19 vs. 0.14	0.14 vs. 0.08 vs. 1.21
Group 4 vs. Group (1+2) vs. Group 3	age	-34.35 vs. -0.04(2) vs. -0.04	-0.17 vs. NaN vs. NaN	0.002 vs. 1.0 vs. 1.0

Table 4.5: Correlation analysis between significant independent variables and target variable.

4.6 NBA Case Studies: Exceptional Players

Similar to discovering NHL exceptional players and their strengths, we apply the same method to NBA datasets to find promising prospects among each group.

Figure 4.4 shows the top players in each group together with their strongest points. In Group 1, the weakest group, players whose strongest points are in *trb_per(total rebounds per game)* and *blk(blocks)* are ranked higher compared to the rest of players, similar to Group 2. Group 3 is a relatively average group. In this group, *Shawn Bradley* is ranked as the greatest player. He is one of the most controversial players in the NBA draft history, well-known for his advantageous height. However, according to the results of our method, his strongest points is in his blocks ability rather than his height. This finding is in accord with his career performance in NBA. *Benoit Benjamin* in Group 4 has the $3rd$ overall pick in his draft year. According to Figure 4.4, he is excellent in scoring points and free throws. Group 5 is the strongest group computed by our model. The most prestigious player *Chris Webber* computed by our model is also a superstar. He is a five-time NBA All-Star, a five-time All-NBA Team member, and NBA Rookie of the Year *(1994)*. His strongest points in pre-draft years are *trb(total rebounds)*, *mp(minutes played)* and *ast(assists)*.

Our model also discovers some players who should receive a better draft order than their actual draft order. *Matt Geiger* in Group 4, was picked 42 in 1992 draft, after *Todd Day(8th), Bryant Stith(13th) Anthony Peeler(15th)*. However, his career PER is 15.2, above those players drafted before him. A more recent case is *Dejuan Blair*, who has the $37th$ overall draft pick in 2009, taken after *Jordan Hill(8th), Ricky Rubio(5th)*, but obtained almost the same career PER as them. In addition, Blair joined two-time The Basketball Tournament defending champion Overseas Elite in summer 2017 and his team, Overseas Elite won its third straight The Basketball Tournament championship with a $86-83$ victory

Group	Top Players	Strongest Points ($\bar{x}$ = group mean)		
1	Paccelis Morlende	trb_per	blk	ft
		18.3 ($\bar{x} = 6.12$)	27 ($\bar{x} = 25$)	138 ($\bar{x} = 121.5$)
	Sani Becirovic	trb_per	blk	ft
		18.3 ($\bar{x} = 6.12$)	27 ($\bar{x} = 25$)	138 ($\bar{x} = 121.5$)
	Cenk Akyol	trb_per	blk	draft_g
		16.4 ($\bar{x} = 6.12$)	26 ($\bar{x} = 25$)	31 ($\bar{x} = 32$)
2	Latavious Williams	blk	trb_per	fg_per
		46 ($\bar{x} = 34$)	7.19 ($\bar{x} = 6.41$)	0.51 ($\bar{x} = 0.50$)
	Ryan Richards	blk	trb_per	fg_per
		46 ($\bar{x} = 34$)	7.19 ($\bar{x} = 6.41$)	0.51 ($\bar{x} = 0.50$)
	Petteri Koponen	blk	fg_per	height
		37 ($\bar{x} = 34$)	0.51 ($\bar{x} = 0.50$)	193 ($\bar{x} = 203$)
3	Shawn Bradley	blk	trb_per	position
		177 ($\bar{x} = 32$)	7.7 ($\bar{x} = 6.6$)	Center
	Kosta Koufos	position	g	trb_per
		Center	37 ($\bar{x} = 32$)	6.7 ($\bar{x} = 6.6$)
	Paulao Prestes	position	trb_per	g
		Center	7.19 ($\bar{x} = 6.6$)	33 ($\bar{x} = 32$)
4	Benoit Benjamin	ft	pts_per	age
		172 ($\bar{x} = 116$)	21.5 ($\bar{x} = 16.86$)	20 ($\bar{x} = 21.38$)
	Hersey Hawkins	ft	pts_per	age
		284 ($\bar{x} = 116$)	36.3 ($\bar{x} = 16.86$)	21 ($\bar{x} = 21.38$)
	Chris Kaman	ft	pts_per	age
		206 ($\bar{x} = 116$)	22.4 ($\bar{x} = 16.86$)	20 ($\bar{x} = 21.38$)
5	Larry Johnson	ft	trb	ast
		162 ($\bar{x} = 122$)	380 ($\bar{x} = 214$)	104 ($\bar{x} = 84$)
	Anfernee Hardaway	trb	ast	mp
		273 ($\bar{x} = 214$)	204 ($\bar{x} = 84$)	1196 ($\bar{x} = 929$)
	Chris Webber	trb	mp	ast
		362 ($\bar{x} = 84$)	1143 ($\bar{x} = 929$)	90 ($\bar{x} = 84$)

Figure 4.13: NBA exceptional players in each group and their strongest points [9].

over Team Challenge ALS on ESPN(`https://en.wikipedia.org/wiki/DeJuan_Blair`). These two underestimated players statistics is shown in Table 4.5.

name	draft_year	draft pick	career PER	predicted PER	comparables(career_per, pick)
Matt Geiger	1992	42	15.2	11.7	Anthony Peeler(12.9, 15th)
Dejuan Blair	2009	37	16.5	17.2	Jordan Hill(16.3, 8th)

Table 4.6: Underestimated players.

Chapter 5

Conclusion

We have proposed building regression model trees for ranking draftees in the NHL & NBA, or other sports, based on a list of player features and performance statistics. The model tree groups players according to the values of discrete features, or learned thresholds for continuous performance statistics. Each leaf node defines a group of players that is assigned its own regression model. Tree models combine the strength of both regression and cohort-based approaches, where player performance is predicted with reference to comparable players. An obvious approach is to use a linear regression tree for predicting dependent variable, like what we did to the NBA datasets. Also, this regression tree method can also be applied to the NHL datasets. However, we found that a linear regression tree performs poorly in NHL due to the zero-inflation problem(many draft picks never play any NHL game). Instead, we introduced the idea of using a logistic regression tree to predict whether a player plays any NHL game within 7 years. Players are ranked according to the model tree probability that they play at least 1 game.

Key findings include the following. 1) The model tree ranking correlates well with the actual success ranking according to the actual number of games played: better than draft order. 2) The model tree can highlight the exceptionally strongest points of draftees that make them stand out compared to the other players in their group.

Tree models are flexible and can be applied to other prediction problems to discover groups of comparable players as well as predictive models. For example, we can predict future NHL success from past NHL success, similar to Wilson [33] who used machine learning models to predict whether a player will play more than 160 games in the NHL after 7 years. Another direction is to apply the model to other sports, for example drafting for the National Football League.

Bibliography

[1] C. Blake and C. Merze. Uci repository of machine learning databases. 1998.

[2] L. Breiman, H. Friedman, J.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. 1984.

[3] D. Cervone, A. D'Amour, L. Bornn, and K. Goldsberry. Pointwise: Prediting points and valuing decisons in real time with nba optical tracking data. In *MIT sloan Sports Analytics Conference*, 2016.

[4] Alexander C.Greene. The success of nba draft picks: Can college career predict nba winners. Master's thesis, St. Cloud State University, 2015.

[5] Dennis Coates and Babatunde Oguntimein. The length and success of nba careers: Does college production predict professional outcomes? *International Journal of Sport Finance*, 5(1), 2008.

[6] Michael E.Schuckers and LLC Statistical Sports Consulting. Draft by numbers: Using data and analytics to improve national hockey league(NHL) player selection. In *MIT sloan Sports Analytics Conference*, 2016.

[7] Rob Found. Goal-based metrics better than shot-based metrics at predicting hockey success. Technical report, 2016.

[8] E. Frank, M. Hall, and I. Witten. *The Weka Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*. Fourth edition, 2016.

[9] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[10] Iain Fyffe. Evaluating central scouting. Technical report, 2011.

[11] M. Hall, E. Frank, G. Homes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: An update. sigkdd explorations. 11:10–18, 2009.

[12] Bill James and Jim Henzler. *Win Shares*. STATS, Inc., 2002.

[13] Eric Joyce and Cameron Lawrence. Blending old and new: How the florida panthers try to predict future performance at the nhl entry draft, 2017.

[14] J.R.QUINLAN. Learning with continuous classes. *World Scientific*, pages 343–348, 1992.

[15] E.H. Kaplan, K. Masri, and J.T. Ryan. A markov game model for hockey: Manpower differential and win probability added. *INFOR: Information Systems and Operational Research*, 52(2):39–50, 2014.

[16] S.B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[17] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Kluwer Academic Publishers*, 2006.

[18] Wei-Yin Loh. *GUIDE User Manual*. University of Wisconsin-Madison, 2017.

[19] Dwight Lutz. A cluster analysis of nba players. In *MIT sloan Sports Analytics Conference*, 2012.

[20] B. Macdonald. An improved adjusted plus-minus statistic for nhl players. In *MIT sloan Sports Analytics Conference*, 2011.

[21] Brian Mazique. Dissecting which position is most important towards winning an nba championship. Technical report, 2012.

[22] Daryl Pregibon. *private communication*. 1989, 1992.

[23] Alan Ryder. Poisson toolbox, a review of the application of the poisson probability distribution in hockey. Technical report, Hockey Analytics, 2004.

[24] Michael Schuckers and James Curro. Total hockey rating (thor): A comprehensive statistical rating of national hockey league forwards and defensemen based upon all on-ice events. In *MIT sloan Sports Analytics Conference*, 2013.

[25] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(379-423), 1948.

[26] Joseph Sill. Improved nba adjusted +/- using regularization and out-of-sample testing. In *MIT sloan Sports Analytics Conference*, 2010.

[27] Nate Silver. *PECOTA 2004: A Look Back and a Look Ahead*, pages 5–10. New York: Workman Publishers, 2004.

[28] A. Thomas, S. Ventural, S. Jensen, and S. Ma. Competing process hazard function model for player ratings in ice hockey. *The Annals of Applied Science*, 7(2):1497–1524, 2013.

[29] A.C. Thomas and Sam Ventura. The highway to war: Defining and calculating the components for wins above replacement. 2015. Available at `https://aphockey.files.wordpress.com/2015/04/sam-war-1.pdf`.

[30] Andrew C. Thomas. The impact of puck possession and location on ice hockey strategy. *Journal of Quantitative Analysis in Sports*, 2(1), 2006.

[31] P. Tingling, K. Masri, and M. Martell. Does order matter? an empirical analysis of nhl draft decisions. *Sport, Business and Management: an International Journal*, (2):155–171, 2011.

[32] Josh W. Draft analytics: Unveiling the prospect cohort success model. Technical report, 2015.

[33] David Wilson. Mining nhl draft data and a new value pick chart. Master's thesis, University of Ottawa, 2016.

Appendix A

Spearman Rank Correlation

Spearman's correlation measures the relevance and direction of monotonic association between two variables [10]. The standard formula for calculating is based on the squared rank differences:

(1) $p = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$, formula for no tied ranks. n = number of ranks, d_i = difference in paired ranks. This is the formula we applied in Table 4.1.

(2) $p = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y}^2)}}$, where x_i = rank of player i according to ranking x, ditto for y_i.

Players who have played zero NHL games are tied when ranked by the number of NHL games; this is the only case of ties. Table A.1 repeats the calculation of Table 4.1 using the Pearson correlation among ranks (2) rather than the squared rank differences (1). With this measure also, the model ranking correlates more highly with actual number of games played than the team draft order.

Training Data NHL Draft Years	Out of Sample Draft Years	Draft Order Pearson Correlation	Tree Model Pearson Correlation
1998, 1999, 2000	2001	0.43	0.69
1998, 1999, 2000	2002	0.45	0.72
2004, 2005, 2006	2007	0.48	0.60
2004, 2005, 2006	2008	0.51	0.58

Table A.1: Pearson Correlation of NHL ranks.

Appendix B

Values of Position_Union in NBA Tree

The values of position_union_1 and position_union_2 in M5 regression tree of NBA are as follows:

Position_Union_1 = (Small Forward, Point Guard and Shooting Guard and Small Forward, Power Forward and Shooting Guard and Small Forward, Power Forward, Small Forward and Point Guard and Shooting Guard, Small Forward and Power Forward, Point Guard, Shooting Guard and Small Forward and Point Guard, Point Guard and Shooting Guard, Small Forward and Shooting Guard, Small Forward and Shooting Guard and Power Forward, Small Forward and Power Forward and Center, Shooting Guard and Power Forward,Power Forward and Small Forward, Shooting Guard and Point Guard,Shooting Guard and Small Forward, Shooting Guard and Small Forward and Power Forward, Center and Power Forward, Power Forward and Center, Point Guard and Small Forward and Shooting Guard,Small Forward and Power Forward and Shooting Guard, Small Forward and Shooting Guard and Point Guard, Center and Small Forward and Power Forward, Power Forward and Center and Small Forward, Small Forward and Center and Power Forward, Center and Power Forward and Small Forward, Shooting Guard and Power Forward and Small Forward)

Position_Union_2 = (Center/Forward, Center and Small Forward, Small Forward and Center, Center, Shooting Guard and Point Guard and Small Forward, Power Forward and Small Forward and Shooting Guard, Shooting Guard, Small Forward, Point Guard and Shooting Guard and Small Forward, Power Forward and Shooting Guard and Small Forward, Power Forward, Small Forward and Point Guard and Shooting Guard, Small Forward and Power Forward, Point Guard, Shooting Guard and Small Forward and Point Guard, Point Guard and Shooting Guard, Small Forward and Shooting Guard, Small Forward and Shooting Guard and Power Forward, Small Forward and Power Forward and Center, Shooting Guard and Power Forward, Power Forward and Small Forward, Shooting Guard and Point Guard, Shooting Guard and Small Forward, Shooting Guard and Small Forward and Power Forward, Center and Power Forward, Power Forward and Center, Point Guard and Small Forward and Shooting Guard, Small Forward and Power Forward and Shooting Guard, Small Forward and Shooting Guard and Point Guard, Center and Small Forward and Power Forward, Power Forward and Center and Small Forward, Small Forward and Center and Power Forward, Center and Power Forward and Small Forward, Shooting Guard and Power Forward and Small Forward)

Appendix C

Code

C.1 Data Collection

C.1.1 NHL Datasets

```python
import csv
import traceback

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import unicodedata
import time
import os
import sys
import os.path

# The following crawling script is built on Galen Liu's scripts
# Many thanks to Galen!


# if use linux server
def find_chrome():
    chromedriver = "/home/cla315/chromedriver"
    os.environ["webdriver.chrome.driver"] = chromedriver
    driver = webdriver.Chrome(chromedriver)
    return driver


def record_dict_value(dict_record, key, value):
    try:
        if value == "":
            dict_record.update({key: "Null"})
        else:
            dict_record.update({key: value})
    except ValueError:
        print "empty value"
        dict_record.update({key: "Null"})
    return dict_record
```

```python


def jump2search(driver, gametype, season):
  if gametype == 2:
    gametype_str = "Regular Season"
  elif gametype == 3:
    gametype_str = "Playoffs"
  print "Now start crawling for Season " + str(season) + "-" + str(season + 1) + " " + gametype_str
  season_str = str(season)+str(season+1)
  player_search_url = "http://www.nhl.com/stats/player?aggregate=0&gameType=" + str(gametype) + "&report=skatersummary&pos=S&reportType=season&seasonFrom=" + season_str + "&seasonTo=" + season_str + "&filter=gamesPlayed,gte,0&sort=playerName"
  driver.get(player_search_url)
  time.sleep(3)
  return driver


def get_store_directory(season, gametype):
  gametype_str=""
  if gametype == 2:
    gametype_str = "RegularSeason"
  elif gametype == 3:
    gametype_str = "Playoffs"

  data_directory = "/home/cla315/work_yeti/NHL_player_stats_season_by_season/" + str(season) +"_" + str(season+1) + "_" + gametype_str + ".txt"
  return data_directory


def crawl_data(txt_file, driver):

  row_path = '//*[@id="stats-page-body"]/div[3]/div/div/div/div/table/tbody[1]'
  num_pages_text =driver.find_element_by_xpath('//*[@id="stats-page-body"]/div[3]/div[2]/div/div[2]/span[2]/span').text
  num_pages_text = unicodedata.normalize('NFKD', num_pages_text).encode('ascii', 'ignore')
  total_num_pages = int(num_pages_text)
  print "Total number of pages to be crawled is " + str(total_num_pages)
  lastPage = True
  print 'check point 1'
  for page_num in range(0, total_num_pages):
    print 'check point 2'
    if lastPage:
      lastPage = False
    else:
      page_pointer = driver.find_element_by_xpath('//*[@id="stats-page-body"]/div[3]/div[2]/div/div[1]/button').click()
    curr_page_num = driver.find_element_by_xpath(
      '//*[@id="stats-page-body"]/div[3]/div[2]/div/div[2]/span[2]/div/input').get_attribute("value")
    curr_page_num = unicodedata.normalize('NFKD', curr_page_num).encode('ascii', 'ignore')
    print "Currently crawling page # " + curr_page_num

    total_num_rows = len(driver.find_elements_by_class_name('rt-tr-group'))
```

```
  print "Total number of rows on page # " + curr_page_num + " is " + str(total_num_rows)          # including blank rows

  row_path = '//*[@id="stats-page-body"]/div[3]/div[1]/div[3]'
  for row_num in range(1, total_num_rows + 1):
    data_record_dict = {}
    # //*[@id="stats-page-body"]/div[3]/div[1]/div[3]/div[1]
    current_row_path = row_path + "/div[" + str(row_num) + "]"

    try:
      id_url_xpath = current_row_path + "/div/div[2]/div/a"
      id_url = driver.find_element_by_xpath(id_url_xpath).get_attribute("href")
    except:
      print "row number is " + str(row_num)
      break

    id_url = unicodedata.normalize('NFKD', id_url).encode('ascii', 'ignore')
    player_id = id_url[(len(id_url)-7):]
    print "player id is " + player_id
    data_record_dict = record_dict_value(data_record_dict, "PlayerID", player_id)

    player_name_xpath = current_row_path + "/div/div[2]/div/a"
    player_name = driver.find_element_by_xpath(player_name_xpath).text
    player_name = unicodedata.normalize('NFKD', player_name).encode('ascii', 'ignore')
    data_record_dict = record_dict_value(data_record_dict, "PlayerName", player_name)

    season_xpath =  current_row_path + "/div/div[3]/div"
    season = driver.find_element_by_xpath(season_xpath).text
    season = unicodedata.normalize('NFKD', season).encode('ascii', 'ignore')
    data_record_dict =  record_dict_value(data_record_dict, "Season", season)

    team_xpath = current_row_path + "/div/div[4]"
    team = driver.find_element_by_xpath(team_xpath).text
    team = unicodedata.normalize('NFKD', team).encode('ascii', 'ignore')
    data_record_dict = record_dict_value(data_record_dict, "Team", team)

    pos_xpath = current_row_path + "/div/div[5]"
    pos = driver.find_element_by_xpath(pos_xpath).text
    pos = unicodedata.normalize('NFKD', pos).encode('ascii', 'ignore')
    data_record_dict = record_dict_value(data_record_dict, "Position",pos)

    gp_xpath = current_row_path + "/div/div[6]"
    gp = driver.find_element_by_xpath(gp_xpath).text
    gp = unicodedata.normalize('NFKD', gp).encode('ascii', 'ignore')
    data_record_dict = record_dict_value(data_record_dict, "GP", gp)

    g_xpath =  current_row_path + "/div/div[7]"
    g = driver.find_element_by_xpath(g_xpath).text
    g = unicodedata.normalize('NFKD', g).encode('ascii', 'ignore')
    data_record_dict = record_dict_value(data_record_dict, "G", g)

    a_xpath = current_row_path + "/div/div[8]"
    a = driver.find_element_by_xpath(a_xpath).text
```

```
        a = unicodedata.normalize('NFKD', a).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "A", a)

        p_xpath = current_row_path + "/div/div[9]"
        p = driver.find_element_by_xpath(p_xpath).text
        p = unicodedata.normalize('NFKD', p).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "P", p)

        pm_xpath = current_row_path + "/div/div[10]"
        pm = driver.find_element_by_xpath(pm_xpath).text
        pm = unicodedata.normalize('NFKD', pm).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "+/-", pm)

        pim_xpath = current_row_path + "/div/div[11]"
        pim = driver.find_element_by_xpath(pim_xpath).text
        pim = unicodedata.normalize('NFKD', pim).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "PIM", pim)

        pgp_xpath = current_row_path + "/div/div[12]/div"
        pgp = driver.find_element_by_xpath(pgp_xpath).text
        pgp = unicodedata.normalize('NFKD', pgp).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "P/GP", pgp)


        ppg_xpath =  current_row_path + "/div/div[13]"
        ppg = driver.find_element_by_xpath(ppg_xpath).text
        ppg = unicodedata.normalize('NFKD', ppg).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "PPG", ppg)

        ppp_xpath =  current_row_path + "/div/div[14]"
        ppp = driver.find_element_by_xpath(ppp_xpath).text
        ppp = unicodedata.normalize('NFKD', ppp).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "PPP", ppp)

        shg_xpath =  current_row_path + "/div/div[15]"
        shg = driver.find_element_by_xpath(shg_xpath).text
        shg = unicodedata.normalize('NFKD', shg).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "SHG", shg)

        shp_xpath =  current_row_path + "/div/div[16]"
        shp = driver.find_element_by_xpath(shp_xpath).text
        shp= unicodedata.normalize('NFKD', shp).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "SHP", shp)

        gwg_xpath =  current_row_path + "/div/div[17]"
        gwg = driver.find_element_by_xpath(gwg_xpath).text
        gwg = unicodedata.normalize('NFKD', gwg).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, "GWG", gwg)

        otg_xpath =  current_row_path + "/div/div[18]"
        otg = driver.find_element_by_xpath(otg_xpath).text
        otg = unicodedata.normalize('NFKD', otg).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "OTG", otg)

        s_xpath =  current_row_path + "/div/div[19]"
        s = driver.find_element_by_xpath(s_xpath).text
        s = unicodedata.normalize('NFKD', s).encode('ascii', 'ignore')
        data_record_dict =  record_dict_value(data_record_dict, "S", s)
```

```
      spercentage_xpath =  current_row_path + "/div/div[20]/div"
      spercentage = driver.find_element_by_xpath(spercentage_xpath).text
      spercentage = unicodedata.normalize('NFKD', spercentage).encode('ascii', 'ignore')
      data_record_dict =  record_dict_value(data_record_dict, "S%", spercentage)

      toigp_xpath =  current_row_path + "/div/div[21]/div"
      toigp = driver.find_element_by_xpath(toigp_xpath).text
      toigp = unicodedata.normalize('NFKD', toigp).encode('ascii', 'ignore')
      data_record_dict =  record_dict_value(data_record_dict, "TOI/GP", toigp)

      shifts_xpath =  current_row_path + "/div/div[22]/div"
      shifts = driver.find_element_by_xpath(shifts_xpath).text
      shifts = unicodedata.normalize('NFKD', shifts).encode('ascii', 'ignore')
      data_record_dict =  record_dict_value(data_record_dict, "Shifts/GP", shifts)

      fow_xpath =  current_row_path + "/div/div[23]/div"
      fow = driver.find_element_by_xpath(fow_xpath).text
      fow = unicodedata.normalize('NFKD', fow).encode('ascii', 'ignore')
      data_record_dict =  record_dict_value(data_record_dict, "FOW%", fow)

      data_record.append(data_record_dict)

    for data_record_line in data_record:
      txt_file.write(str(data_record_line))
      txt_file.write("\n")


def start_crawl(gameType, season):
  chrome_driver = find_chrome()
  search_page_driver = jump2search(chrome_driver, gameType, season)
      # jump2search(driver, gameType, season_num)
      # gameType = 2 for regular seasons, = 3 for playoffs
  data_directory = get_store_directory(season, gameType )
  if os.path.exists(data_directory):
    with open(data_directory, "a") as txt_file:
      crawl_data(txt_file, search_page_driver)
  else:
    with open(data_directory, "w") as txt_file:
      crawl_data(txt_file, search_page_driver)
  chrome_driver.close()



if __name__ == '__main__':
  for season in range (2014, 2015):
    for gameType in range (3,4):
      start_crawl(gameType, season)

```

Listing C.1: Crawl NHL season data from `www.nhl.com`

```
import csv
import traceback
import shutil
```

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import unicodedata
import time
import os
import sys
import os.path
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.chrome.options import Options

# if use linux server
def find_chrome():
  chromedriver = "/home/cla315/chromedriver"
  os.environ["webdriver.chrome.driver"] = chromedriver
  chrome_options = Options()
  chrome_options.add_experimental_option("prefs", {'profile.managed_default_content_settings.javascript': 2,
                            'profile.managed_default_content_settings.images': 2 })
  # chrome_options.add_experimental_option("prefs", {'profile.managed_default_content_settings.images': 2})
  #chrome_options.add_experimental_option("prefs", {'profile.managed_default_content_settings.extensions': 2})
  driver = webdriver.Chrome(chromedriver, chrome_options=chrome_options)
  return driver

def jump2search(driver, playerUrl):
  # driver.implicitly_wait(10)
  t = time.time()
  try:
    driver.get(playerUrl)
    driver.implicitly_wait(300)
  except TimeoutException:
    print 'Couldn\'t load this page.'
    print "Time consuming: " + str(t)
  print "Time consuming: " + str(t)
  return driver

def get_store_directory(draftYr):
  data_directory = "/home/cla315/work_yeti/elite_prospect/playerstats_txtfiles/players_stats_" + draftYr + ".txt"
  return data_directory

def record_dict_value(dict_record, key, value):
  try:
    if value == "":
      dict_record.update({key: "Null"})
    else:
      dict_record.update({key: value})
  except ValueError:
    print "empty value"
    dict_record.update({key: "Null"})
  return dict_record


```

```python
def crawl_data(txt_file, driver):
  data_record = []
  demographic_dict = {}
  position_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[3]/td[2]'
  position = driver.find_element_by_xpath(position_xpath).text
  position = unicodedata.normalize('NFKD', position).encode('ascii', 'ignore')
  if position == 'G':
    print 'This is a goal tender, jump to the next player.'
    return
  demographic_dict = record_dict_value(demographic_dict, "Position", position)

  elite_id_xpath = '/html/head/meta[6]'
  elite_id = driver.find_element_by_xpath(elite_id_xpath).get_attribute("content")
  elite_id = unicodedata.normalize('NFKD', elite_id).encode('ascii', 'ignore')
  elite_id = elite_id.split('player=')
  elite_id = elite_id[1]
  demographic_dict = record_dict_value(demographic_dict, "eliteId", elite_id)

  name_xpath = '//*[@id="fontHeader"]'
  name = driver.find_element_by_xpath(name_xpath).text
  name = unicodedata.normalize('NFKD', name).encode('ascii', 'ignore')
  print 'Player name is ' + name
  demographic_dict = record_dict_value(demographic_dict, "PlayerName", name)

  born_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[1]/td[2]/a'
  born = driver.find_element_by_xpath(born_xpath).get_attribute('href')
  born = unicodedata.normalize('NFKD', born).encode('ascii', 'ignore')
  born = born.split('Birthdate=')
  born = born[1].split('&')
  born = born[0]
  demographic_dict = record_dict_value(demographic_dict, "BirthDate", born)

  try:
    birthplace_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[1]/td[4]/a'
    birthplace = driver.find_element_by_xpath(birthplace_xpath).text
    birthplace = unicodedata.normalize('NFKD', birthplace).encode('ascii', 'ignore')
  except:
    birthplace = "Null"
    print "Null birthplace found."
  demographic_dict = record_dict_value(demographic_dict, "Birthplace", birthplace)

  nation_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[2]/td[4]/a'
  nation = driver.find_element_by_xpath(nation_xpath).text
  nation = unicodedata.normalize('NFKD', nation).encode('ascii', 'ignore')
  demographic_dict = record_dict_value(demographic_dict, "Nation", nation)

  shoots_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[3]/td[4]'
  shoots = driver.find_element_by_xpath(shoots_xpath).text
  shoots = unicodedata.normalize('NFKD', shoots).encode('ascii', 'ignore')
  demographic_dict = record_dict_value(demographic_dict, "Shoots", shoots)
```

```

    height_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[4]/td[2]'
    height = driver.find_element_by_xpath(height_xpath).text
    height = unicodedata.normalize('NFKD', height).encode('ascii', 'ignore')
    demographic_dict = record_dict_value(demographic_dict, "Height", height)

    weight_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[4]/td[4]'
    weight = driver.find_element_by_xpath(weight_xpath).text
    weight = unicodedata.normalize('NFKD', weight).encode('ascii', 'ignore')
    demographic_dict = record_dict_value(demographic_dict, "Weight", weight)

    draftYear_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[6]/td[2]/a'
    draftYear = driver.find_element_by_xpath(draftYear_xpath).text
    draftYear = unicodedata.normalize('NFKD', draftYear).encode('ascii', 'ignore')
    draftYear = draftYear[:4]
   # print 'correct year? ' + draftYear
    demographic_dict = record_dict_value(demographic_dict, "draftYear", draftYear)

    draftRound_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[6]/td[2]/a/strong[1]'
    draftRound = driver.find_element_by_xpath(draftRound_xpath).text
    draftRound = unicodedata.normalize('NFKD', draftRound).encode('ascii', 'ignore')
    demographic_dict = record_dict_value(demographic_dict, "draftRound", draftRound)

    overall_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[6]/td[2]/a/strong[2]'
    overall = driver.find_element_by_xpath(overall_xpath).text
    overall = unicodedata.normalize('NFKD', overall).encode('ascii', 'ignore')
    demographic_dict = record_dict_value(demographic_dict, "Overall", overall)

    overallBy_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[6]/td[2]/a/b'
    overallBy = driver.find_element_by_xpath(overallBy_xpath).text
    overallBy = unicodedata.normalize('NFKD', overallBy).encode('ascii', 'ignore')
    demographic_dict = record_dict_value(demographic_dict, "overallBy", overallBy)


    label_xpath = '//*[@id="fontSmall"]'
    label = driver.find_element_by_xpath(label_xpath).text
    label = unicodedata.normalize('NFKD', label).encode('ascii', 'ignore')
    if label == "YOUTH TEAM":
      youth_team_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/p/table[2]/tbody/tr/td[1]/table/tbody/tr[5]/td[2]/a'
      youth_team = driver.find_element_by_xpath(youth_team_xpath).text
      youth_team = unicodedata.normalize('NFKD', youth_team).encode('ascii', 'ignore')
    else:
      youth_team = 'Null'
```

```
  demographic_dict = record_dict_value(demographic_dict, "youthTeam", youth_team)
  print 'Demographic info has been read.'

  # the following code find all field names for a season
  field_name_dict = {1:'Team', 2:'League', 3:'Reg_GP', 4:'Reg_G', 5:'Reg_A', 6:'Reg_TP', 7:'Reg_PIM',
           8:'Reg_PlusMinus', 10: 'Post', 11:'Play_GP', 12:'Play_G', 13:'Play_A',
           14:'Play_TP', 15:'Play_PIM', 16:'Play_PlusMinus'}
  # print 'Field names are: ' + str(field_name_dict)

  table_xpath = '/html/body/div[2]/table[3]/tbody/tr/td[5]/table[1]/tbody'
  table = driver.find_element_by_xpath(table_xpath)
  # find all row elements in the table
  all_rows = table.find_elements_by_tag_name('tr')
  field_name_row = True
  last_season = ""
  for row in all_rows:
    data_record_dict = dict(demographic_dict)
    # skip the first row in the table: the field names
    if field_name_row:
      field_name_row = False
      continue
    # find all columns elements for each row
    cols = row.find_elements_by_tag_name('td')
    season = cols[0].text
    season = unicodedata.normalize('NFKD', season).encode('ascii', 'ignore')
    # print "what's in the blank????????" + season + '????????'
    if season != " ":
      last_season = season
      print "Current season is " + season
    else:
      season = last_season
      print "The season of this row is blank. Set season to be last season " + season

    yearList = season.split("-")
    yearStr = yearList[0]
    if int(yearStr) >= int(draftYear):
      print 'Season is out of range, go to the next player.'
      break
    if int(yearStr) < int(draftYear)-1:
      print 'Not the target year yet, keep moving down.'
      continue
    # now it's the season we want to record
    data_record_dict = record_dict_value(data_record_dict, 'Season', season)
    try:
      for col_num in field_name_dict.keys():
        field_value = cols[col_num].text
        field_value = unicodedata.normalize('NFKD', field_value).encode('ascii', 'ignore')
        data_record_dict = record_dict_value(data_record_dict, field_name_dict[col_num], field_value)
      print "Season record to be saved is: " + str(data_record_dict)
      data_record.append(data_record_dict)
    except IndexError:
      break
```

```python

  for data_record_line in data_record:
    # print 'Is the record written to txt files????'
    txt_file.write(str(data_record_line))
    txt_file.write("\n")

def start_crawl(inputDir, moveToDir):
  fileNameList = []
  for file in os.listdir(inputDir):
    if file.endswith(".csv"):
      fileNameList.append(file)
  print "The following csv files will be imported to the database." +
    fileNameList[0]

  for fileName in fileNameList:
    draftYear = fileName[-8:-4]

    print 'Draft year is ' + draftYear
    data_directory = get_store_directory(draftYear)

    with open(inputDir + "/" + fileName, 'r') as inputFile:
      csv_data = csv.reader(inputFile)
      # the following code avoids importing headers/1st row in each csv file
      firstLine = True
      for row in csv_data:
        if firstLine:
          firstLine = False
          continue
        player_url = row[1]
        chrome_driver = find_chrome()
        driver = jump2search(chrome_driver, player_url)

        if os.path.exists(data_directory):
          with open(data_directory, "a") as txt_file:
            crawl_data(txt_file, driver)
        else:
          with open(data_directory, "w") as txt_file:
            crawl_data(txt_file, driver)
        chrome_driver.close()
    shutil.move(inputDir + "/" + fileName, moveToDir + "/" + fileName)

if __name__ == '__main__':
  input_dir = "/home/cla315/work_yeti/elite_prospect/url_csv_files"
  # input_dir = "/home/cla315/work_yeti/elite_prospect/test_folder"
  moveto_dir = "/home/cla315/work_yeti/elite_prospect/url_csv_files_old"
  # moveto_dir = "/home/cla315/work_yeti/elite_prospect/test_folder"
  start_crawl(input_dir, moveto_dir)



```

Listing C.2: Crawl pre-draft NHL prospects data from `www.eliteprospects.com`

C.1.2 NBA Datasets

```python
import re
import requests
from bs4 import BeautifulSoup, Comment, Tag
```

```python
import urllib.parse

from commit2db import MysqlConnection

YEAR_URL = 'https://www.basketball-reference.com/draft/NBA_{year}.html'
HEADERS = {
  'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2171.95 Safari/537.36'}


def regex_wrapper(found: list) -> str:
  return found[0] if found else ''


def get_player_info(soup: Tag) -> dict:
  meta_soup = soup.find(id='meta')
  player_data = {}
  player_data['name'] = meta_soup.h1.get_text()

  all_paragraph = meta_soup.select('div > p')

  # The original html is messy, sorry for the regex
  meta_data = meta_soup.get_text().replace('\n', ' ')
  height, weight = regex_wrapper(re.findall(r'\((\d+)cm.*?(\d+)kg', meta_data))
  player_data['position'] = regex_wrapper(re.findall(r'Position\:?\s+(.*?)\s+', meta_data))
  player_data['shoots'] = regex_wrapper(re.findall(r'Shoots\:?\s+(\w+\s?\w+)', meta_data))
  player_data['height'] = int(height)
  player_data['weight'] = int(weight)

  # College data is hard to use regex...
  if 'College' in meta_data:
    all_links = [x for x in meta_soup.find_all('a')]
    result = list(filter(lambda x: 'college' in x['href'], all_links))[0]
    player_data['college'] = result.get_text()

  # some of them just don't have following data,
  # so...
  try:
    player_data['born'] = meta_soup.find(id='necro-birth')['data-birth']
    player_data['team'] = regex_wrapper(re.findall(r'Team\:?\s+(\w+\s?\w+)', meta_data))
    # Easily broke here, pay attention to index
    player_data['nba_debut'] = all_paragraph[-2].a.get_text()
  except Exception:
    print("He doesn't have team or nba_debut...")
  return player_data


def str2float(string: str, default=None):
  try:
    return float(string)
  except ValueError:
    if default == None:
      return string
    else:
```

```
    return default


def get_career_data(soup: Tag) -> dict:
  try:
    career_data = soup.find('div', {'class': 'stats_pullout'})
    career_data = career_data.select('div > p')[2:]
    career_data = [str2float(x.get_text(), 0) for x in career_data]
    player_career = {}
    player_career['G'] = career_data[1]
    player_career['PTS'] = career_data[3]
    player_career['TRB'] = career_data[5]
    player_career['AST'] = career_data[7]
    player_career['FG'] = career_data[9]
    player_career['FG3'] = career_data[11]
    player_career['FT'] = career_data[13]
    player_career['eFG'] = career_data[15]
    player_career['PER'] = career_data[17]
    player_career['WS'] = career_data[19]
  except Exception:
    raise LookupError
  return player_career


def get_college_data(soup: Tag) -> dict:
  # I don't understand why they first comment the section
  # then uncomment it in runtime. Reduce rendering time?
  all_comments = soup.findAll(text=lambda x: isinstance(x, Comment))
  try:
    comment = list(filter(lambda x: 'College Table' in x, all_comments))[0]
  except IndexError:
    raise LookupError
  comment = BeautifulSoup(comment, 'html.parser')
  career_tr = comment.select('tfoot > tr')[0]

  def get_each_season(tr_soup: Tag):
    season_data = {}
    season_data['season'] = tr_soup.th.get_text()
    all_td = [str2float(x.get_text(), 0) for x in tr_soup.findAll('td')][-7:]
    all_columns = ['FG', '3P', 'FT', 'MP', 'PTS', 'TRB', 'AST']
    for index, column in enumerate(all_columns):
      season_data[column] = all_td[index]
    return season_data

  career_tr = get_each_season(career_tr)
  return career_tr


def get_person(url: str) -> tuple:
  html_page = requests.get(url, headers=HEADERS)
  html_page = html_page.content.decode('utf-8')
  drink_soup = BeautifulSoup(html_page, 'html.parser')
  player_info = get_player_info(drink_soup)
  try:
    career_data = get_career_data(drink_soup)
  except LookupError:
    career_data = {}
  try:
```

```python
    college_data = get_college_data(drink_soup)
  except LookupError:
    college_data = {}
  return player_info, career_data, college_data


def get_person_list_by_year(year: int) -> list:
  url = YEAR_URL.format(year=year)
  year_html = requests.get(url, headers=HEADERS).content.decode('utf-8')
  year_soup = BeautifulSoup(year_html, 'html.parser')
  year_soup = year_soup.find(id='stats')
  player_list = year_soup.select('tbody > tr')

  def handle_one_player(player: Tag) -> tuple:
    try:
      pk = player.find('td', {'data-stat': 'pick_overall'}).get_text()
      url = player.find('td', {'data-stat': 'player'}).a['href']
    except:
      return ()
    return urllib.parse.urljoin(YEAR_URL, url), int(pk)

  player_list = [handle_one_player(x) for x in player_list]
  return player_list


def test(url: str):
  mysql = MysqlConnection()
  try:
    player_info, career_data, college_data = get_person(url)
  except IndexError:
    print("No college data")
    return
  player_info['draft_year'] = 9999
  player_info['ID'] = 999395
  print(player_info, career_data, college_data)
  mysql.save_to_db(player_info, career_data, college_data)


if __name__ == '__main__':
  # test('https://www.basketball-reference.com/players/l/ledori01.html')
  mysql = MysqlConnection()
  for draft_year in range(2012, 2017):
    person_list = get_person_list_by_year(draft_year)
    person_list = filter(None, person_list)
    for person_url, pk in person_list:
      player_info, career_data, college_data = get_person(person_url)
      player_info['draft_year'] = draft_year
      player_info['pk'] = pk
      player_info['ID'] = draft_year * 100 + pk
      print(person_url)
      mysql.save_to_db(player_info, career_data, college_data)
```

Listing C.3: Crawl career and pre-draft NBA drafted players data from `https://www.basketball-reference.com/draft/`

C.2 Strongest Points Calculation

```
  import csv
import pandas as pd
import numpy as np

DraftAge_norm_avg = [0.033096926713948, 0.043478260869565216,
    0.051051051051051066, 0.08653846153846147, 0.04569892473118278,
    0.11111111111111112]
Weight_norm_avg = [0.46071202864967375, 0.4619027120103317,
    0.43403799839443397, 0.4305026656511804,
    0.4408335994889812,0.4435643564356436]
CSS_rank_norm_avg = [0.02179135209334248, 0.7012622720897614,
    0.5062663469921534, 0.5087624069478907, 0.409176638917794,
    0.5266129032258065]
rs_A_norm_avg = [0.2916827852998066, 0.041106719367588924, 0.2308353808353808,
     0.25480769230769235, 0.26370967741935497, 0.43090909090909085]
rs_P_norm_avg = [0.28802625622453604, 0.03399629972247919,
    0.22232604945370898, 0.251943535188216, 0.2510724090597117,
    0.375531914893617]
rs_GP_norm_avg = [0.5553191489361703, 0.4043478260869566, 0.6372972972972974,
    0.463076923076923,0.6346774193548387, 0.67]
rs_G_norm_avg = [0.2828696126568466, 0.03623188405797101, 0.204088704088704,
    0.2480276134122287, 0.23325062034739463, 0.29743589743589743]
rs_PIM_norm_avg = [0.21537125488493275, 0.12658976634131913,
    0.26112336826622534, 0.1817765567765567, 0.22939982444590737,
    0.21904761904761907]
rs_PlusMinus_norm_avg = [0.4263127073980092, 0.3310729956122856,
    0.26270766179023064, 0.35779816513761553, 0.4702574726250369,
    0.5688073394495413]
po_P_norm_avg = [0.16188714153561518, 0.021739130434782605,
    0.06022326674500589, 0.09824414715719063, 0.10483870967741934,
    0.4782608695652174]
po_PIM_norm_avg = [0.13235294117647056, 0.0588235294117647,
    0.05007949125596183, 0.09968891402714929, 0.12737191650853882,
    0.29411764705882354]
po_PlusMinus_norm_avg = [0.3154805575935436, 0.30884557721139444,
    0.27912395153774466, 0.309018567639257, 0.3145161290322579,
    0.5931034482758621]
po_A_norm_avg = [0.14680851063829783, 0.023188405797101 45,
    0.05855855855855854, 0.09198717948717947, 0.10026881720430098,
    0.5133333333333333]
po_GP_norm_avg = [0.26559060895084374, 0.1904047976011994,
    0.14725069897483686, 0.18965517241379315, 0.264460511679644,
    0.6620689655172415]
Height_norm_avg = [0.5784574468085106, 0.6086956521739131, 0.5717905405405406,
     0.5612980769230769, 0.5594758064516129, 0.4875]
country_EURO_avg = [0.5106, 0.1304, 0.1486, 0.2404, 0.1371, 0.2000]
country_USA_avg = [0.1489, 0.3043, 0.1757, 0.4135, 0.2661, 0]
country_CAN_avg = [0.3404, 0.5652, 0.6757, 0.3462, 0.5968, 0.8]
position_R_avg = [0.1915, 0.1739, 0.1216, 0.1346, 0.1613, 0.2]
position_D_avg = [0.3617, 0.6087, 0.3243, 0.3558, 0.3468, 0.4]
position_L_avg = [0.1702, 0.1304, 0.1216, 0.2212, 0.1452, 0.2]

weights_DraftAge = [0.6521969619, 0.6521969619, 1.2673609724, 1.2673609724,
    -1.1198806299, 0.0975778883]
weights_Weight = [3.0647516341, 1.1902369165, 2.1613141404, 0.2293666258,
    1.1902369165, 1.1902369165]
```

```python
weights_CSS_rank = [-19.0327179101, -1.7003217418, -1.9138640157, -1.4879418811, -0.3503937449, -0.3503937449]
weights_rs_A = [1.416228717, 1.416228717, 0.4239688038, 0.4239688038, 0.4239688038, 0.4239688038]
weights_rs_P = [2.5182072847, 9.2361288366, 0, -0.5796943481, 0.5370008872, 0]
weights_country_EURO = [-0.2515058744, -0.3823693577, -1.2307595534, 0.463850402, -0.9510946135, -8.8551596038]
weights_country_USA = [0, -0.2452964509, 0.8069462064, 0, 0, 0]
weights_country_CAN = [0.7372739476, 0, 0, 0, 0, 0]
weights_rs_GP = [0, 0.623606065, 1.5673201838, 1.0477475885, 1.0504894229, 1.0504894229]
weights_rs_PIM = [0, -0.4704279537, -0.8031514348, -0.8031514348, -0.8031514348, -0.8031514348]
weights_rs_PlusMinus = [0, -0.516339445, 2.9754056872, 2.9754056872, -0.8075548697, -0.8075548697]
weights_rs_G = [0, -0.4156266673, -0.4156266673, -0.4156266673, 0.1686151641, -0.4156266673]
weights_po_A = [2.5818151743, 0, 0, 0, 0, 0]
weights_po_P = [0, 0, -2.6843997425, -0.2898101074, 0, 0]
weights_po_GP = [0, -0.6961231358, 0, 0, 0.3212562576, 1.089093825]
weights_po_PIM = [0, 0.7606931178, 0.7606931178, 0.7606931178, 1.4663079409, 1.4663079409]
weights_po_PlusMinus = [4.5578802919 , 0, -2.447755103, -2.447755103, 0, 0]
weights_position_R = [0, 0.6514414089, 0, 0, 0.1937918628, 0]
weights_Height = [-0.9108602568, -1.6148625242, -2.2083451086, -2.2083451086, 0.1943101707, -0.6618192176]
weights_position_D = [-0.3626622956, -0.2475813803, 0.105873252, 0.105873252, 0.310508047, 0.105873252]
weights_position_L = [0, 0, 0.4271360663, -0.0688557505, 0.1239741776, 0.1239741776]

with open('Desktop/output_with_diff_07_08.csv', 'rb') as csvfile:
    d_reader = csv.DictReader(csvfile)
    res = []
    for row in d_reader:
        res_item = (weights_DraftAge[int(row['LeafNode'])-1] * DraftAge_norm_avg[int(row['LeafNode'])-1] +
                    weights_Weight[int(row['LeafNode'])-1] * Weight_norm_avg[int(row['LeafNode'])-1] +
                    weights_CSS_rank[int(row['LeafNode'])-1] * CSS_rank_norm_avg[int(row['LeafNode'])-1] +
                    weights_rs_A[int(row['LeafNode'])-1] * rs_A_norm_avg[int(row['LeafNode'])-1] +
                    weights_rs_P[int(row['LeafNode'])-1] * rs_P_norm_avg[int(row['LeafNode'])-1] +
                    weights_country_EURO[int(row['LeafNode'])-1] * country_EURO_avg[int(row['LeafNode'])-1] +
                    weights_country_USA[int(row['LeafNode'])-1] * country_USA_avg[int(row['LeafNode'])-1] +
                    weights_country_CAN[int(row['LeafNode'])-1] * country_CAN_avg[int(row['LeafNode'])-1] +
                    weights_rs_GP[int(row['LeafNode'])-1] * rs_GP_norm_avg[int(row['LeafNode'])-1] +
                    weights_rs_PIM[int(row['LeafNode'])-1] * rs_PIM_norm_avg[int(row['LeafNode'])-1] +
                    weights_rs_PlusMinus[int(row['LeafNode'])-1] * rs_PlusMinus_norm_avg[int(row['LeafNode'])-1] +
```

```python
                        weights_rs_G[int(row['LeafNode'])-1] * rs_G_norm_avg[int(row['LeafNode'])-1] +
                        weights_po_A[int(row['LeafNode'])-1] * po_A_norm_avg[int(row['LeafNode'])-1] +
                        weights_po_P[int(row['LeafNode'])-1] * po_P_norm_avg[int(row['LeafNode'])-1] +
                        weights_po_GP[int(row['LeafNode'])-1] * po_GP_norm_avg[int(row['LeafNode'])-1] +
                        weights_po_PIM[int(row['LeafNode'])-1] * po_PIM_norm_avg[int(row['LeafNode'])-1] +
                        weights_po_PlusMinus[int(row['LeafNode'])-1] * po_PlusMinus_norm_avg[int(row['LeafNode'])-1] +
                        weights_position_R[int(row['LeafNode'])-1] * position_R_avg[int(row['LeafNode'])-1] +
                        weights_Height[int(row['LeafNode'])-1] * Height_norm_avg[int(row['LeafNode'])-1] +
                        weights_position_D[int(row['LeafNode'])-1] * position_D_avg[int(row['LeafNode'])-1] +
                        weights_position_L[int(row['LeafNode'])-1] * position_L_avg[int(row['LeafNode'])-1])
        print row['LeafNode']
        print res_item
        res.append(res_item)

with open('Desktop/output_with_diff_07_08.csv', 'rb') as input, open('Desktop/output_07_08_weighted_mean.csv', 'wb') as output:
    reader = csv.reader(input, delimiter = ',')
    writer = csv.writer(output, delimiter = ',')

    row = next(reader)  # read title line
    row.append('weighted_mean')
    writer.writerow(row)  # write enhanced title line

    it_1 = res.__iter__()

    for row in reader:
        if row:  # avoid empty lines that usually lurk undetected at the end of the files
            try:
                row.append(next(it_1))
            except StopIteration:
                row.append("N/A")     # not enough results: pad with N/A
            writer.writerow(row)


```

Listing C.4: Calculating strongest points for exceptional players in NHL.*Similar code is applied to NBA prospects, not shown here*