Project Slam Dunc

GitHub Repository

https://github.com/pvacek/160-Team-Duncan

Group Members

Patrick Vacek | ID: 999676176
Chirag Kashyap | ID: 998388067
Edie Espejo | ID: 998874416

Table of Contents

Abstract

Basketball is the third most popular sport in the United States.[1] For regular season games televised by ESPN and ABC in 2016-2017, average viewership was 1.9 million.[2] Because of the sport's popularity and revenue, teams were able to install a system of cameras that measures basketball motion data (SportVu). This SportVu system is in all basketball arenas and measures the real-time position of the basketball and players on the court 25 times per second for every game.[3] This spatial data was originally available to the public. However, after the NBA website received an enormous amount of API requests, it was cut off from public consumption, only available to teams and some media organizations.

To our knowledge, there exists no popular machine learning method to classify basketball plays based on the motion data. Therefore, we have attempted to fill this gap by classifying two popular basketball plays: shots and passes. The classification methods we propose in this project will help automate the collection of game statistics and the verification of boxscores.

Our end result was two-pronged. When we applied traditional machine learning methods to classify missed and made shots, we ended up with a hard accuracy rate of 70%. For passes, we developed a novel method, which we call the Gradient Algorithm. For a sample of 30 games, we were able to classify passes properly at 50% or better for 10 of these. The other 20 seemed to not properly work for issues that we did not have time to resolve.

Stakeholders

The ability to classify popular basketball plays can be of use to both teams and casual viewers. This project will particularly aim to be of assistance to the data experts in their development of future ball movement algorithms.

Scorekeepers

An NBA scorekeeper records statistics as the game is happening and helps resolve disputed calls [4]. With the use of a basketball movement classification algorithm based on ball motion data, scorekeepers may be able to verify their real-time statistical recordings with the results from the SportVU ball motion data. Although future iterations of machine learning algorithms similar to ours might make their job obsolete, current iterations would greatly help referees and scorekeepers make the right call.

General Audience

The general basketball audience may also find the ability to classify basketball plays interesting via use of a web application. A potential web application could allow the sports data layman to choose a game of interest via a dropdown list, then follow a guided exploration of the ball movement for that game. A web application such as this may be created in the future with continued research on basketball motion data to educate and entertain basketball fans.

In the future, another possible use of this data would be to automatically create NBA boxscores without the use of a scorekeeper. These boxscores could expand their available statistics to passes and dribbles, and enhance the casual viewer experience by giving more detailed descriptions of the game. Many sports media companies would be interested in such a project.

NBA Teams

Although not much is known about in-house machine learning methods that NBA teams use, if they manually input raw data and cut game film there could be machine learning methods that could automate this process. Automating everyday tasks could save NBA teams time, money, and allow coaches and players to watch annotated game film minutes after each game. We would be surprised if many NBA teams are not already doing what we proposed at an even more accurate and expansive level, making them not the primary stakeholder.

Data Acquisition

Project Slam Dunc used 15 gigabytes of SportVU ball motion data for inference. These ball motion data were uploaded on Github by Neil Johnson and covers 636 games from October 2015 to January 2016. The auxiliary data were accessed via the NBA API using the `nbapy` package in Python. Using the nbapy package, we wrote a script called `extractaux.py` to acquire the play-by-play game logs and a list of shots for every game in our dataset.

Data Preprocessing

The basketball motion data acquired through Johnson's Github page required multiple steps of preprocessing. The original SportVU data was in 7-zip format and uploaded to a Github repository. To download these data, we used the script called `PTD_dump.R`. In this script were functions to webscrape links of all the files listed in the repository, extract the data from the links. Then, we saved the 636 game motion 7-zip files to an external hard drive. The 7-zip files were on average about 5 megabytes, but there existed some files that were less than 1 kilobyte. We attributed these smaller files to files that were not properly uploaded to Github and considered them to be defective. We subsequently tossed these files before continuing on with data preprocessing.

The 7-zip files were all extracted into JSON files using the `os` module in Python in the `7ztoJSON.py` script. This increased the size of our data considerably to about 70-100 megabytes per file, making the entire dataset around 50 gigabytes.

The average speed to read in one of our 70-100 megabyte JSON ball motion data files into Python was about 3-5 minutes. We had to find a more time efficient way. We theorized a way to convert the JSON files to a CSV format using Bash, a UNIX shell. After initial efforts were fruitless, we found a Bash method called jq that was made to parse JSON files efficiently. Using `jsonTOtxt.sh` to call `jsonTOtxt.py`, we converted all of the JSON files to text files that contained all of the necessary data. These scripts used the architecture of a JSON file to its advantage; it parsed through the extraneous data to only keep the moments of events (the spatial data). This conversion reduced the average file size from around 85 megabytes to around 50 megabytes.

To further format the data in a way convenient for analyses, we created the scripts `txtTOcsv.sh` that called upon `txtTOcsv.py`. This script removed and replaced the remaining JSON formatting such as brackets and commas, removed duplicate snapshots, and then saved the text file in a CSV format. This conversion further reduced the average file size to around 25 megabytes, making our final dataset around 14 gigabytes.The average read time was less than 5 seconds. After having removed 7 defective files, the final product of the data preprocessing pipeline were 629 ball motion data files.

Methods

Project Slam Dunc covers two key classification goals in which we employed three popular machine learning algorithms. We will first begin by describing the types of Machine Learning approaches used in this project.

Functional Data Analysis

In order to classify made or missed shots, we wanted to extract the useful information from the shot heights. Each shot can be measured as functional data, because it takes on a curved form with variation. We hypothesize that shots form an order 4 polynomial, so we used the `fda` package in R to fit splines to the data. Therefore our shots would be approximated of the following form:

$$z(t) \approx c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

This would be our initial, non-technical estimate of approximating the data.

Moving onto a more technical approach, we decided to use Functional Principal Component Analysis, or FPCA for short. We applied FPCA by using the `fdapace` package, a product of our own Statistics department at UC Davis. The motivation behind FPCA is that each function can be decomposed into the mean function plus an infinite sum of eigenfunctions, weighted by scalar principal components. In practice, we would like to keep a finite amount, so our functional data takes the following form:

$$X(t) \approx X_m(t) = \mu(t) + \sum_{k=1}^{m} \xi_k \varphi_k(t),$$

The decision to use FPCA is due to the fact that if functions have differing characteristics, they will have different principal components on average. So we have a hypothesis that $\xi_{i|0} \neq \xi_{i|1}$ for any of the eigenfunctions we use, with 0 being the missed shot class and 1 being the made shot class. In practice, we decided to keep the first 4 components, as they explained over 99% of the functional variation, so our equation took the following form:

$$z(t) \approx z_4(t) = \mu(t) + \xi_1\varphi_1(t) + \xi_2\varphi_2(t) + \xi_3\varphi_3(t) + \xi_4\varphi_4(t)$$

Support Vector Machine

Support Vector Machine (SVM) is a parametric machine learning algorithm. The boundaries created by SVM are created by maximizing the distance of a projection of the distinct classes. Distance for SVM may be described by different kernels for computational efficiency. In R^2 , SVM separates different classes by boundaries based on line segments and curves[5]. In R^3 , SVM creates boundaries with planes to classify different classes. For R^4 and above, SVM may use boundaries based on hyperplanes to classify multiple classes.

Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is also a parametric machine learning method that separated classes by a surface, but this time a quadratic surface.

Boundaries in QDA are based on finding the class *k* which maximizes the quadratic discriminant function below [6].

$$\delta_k(x) = -\frac{1}{2}log|\Sigma k - \frac{1}{2}(x-\mu_k)T\Sigma^{-1}{}_k\,(x-\mu_k) + log\pi_k$$

In R^2, QDA creates parabolas. In higher spaces, QDA creates the canonical higher dimension quadratic surfaces.

Random Forests

Random Forests is an ensemble machine learning algorithm that creates many Decision Trees and uses the mode of these trees to make decisions on which class an observation belongs to [7]. Decision Trees are a non-parametric machine learning method that splits up the classification process into binary problems based on features in a dataset [8]. Random Forests, based on the machine learning method called "bagging", solve the problem of overfitting in Decision Trees [7].

Classification

Classifying Shots

In this first portion of our project, we wished to classify whether or not the basketball was being shot by a player given the ball motion data.

Algorithm

Our shot finding algorithm (`Shots Algorithm.ipynb`) works by finding all of the parabolas within the Z-axis of the ball whenever it is in play. In order to do this, we created a function to get all of the relative minima from a smoothed version of the ball's Z-axis, so that going from one minima to the next is a parabola. Conveniently, we obtained X-axis and Y-axis estimated coordinates for each shot from the NBA API via the `nbapy` package.

From there, we tried to match the estimated coordinates to coordinates obtained from the motion data. We also predicted the Y value of the shot when it got close to the rim by calculating the slope and intercept between two snapshots. Taking a combination of the two methods proved to have the greatest accuracy. In order to further improve accuracy, we implemented a vote space system by taking the closest 10 snapshots and matching those snapshots to parabolas. The parabola with the most snapshots would be our “shot” for each shot obtained from the NBA API. From there, we filtered our shot database by only accepting shots whose maximum Z-value was above seven feet. After filtering and accounting for missing data, we ended up with close to 77,000 shots. In our dataset, all X/Y/Z coordinates were interpolated into 50 X/Y/Z variables so that classification could be easily done.

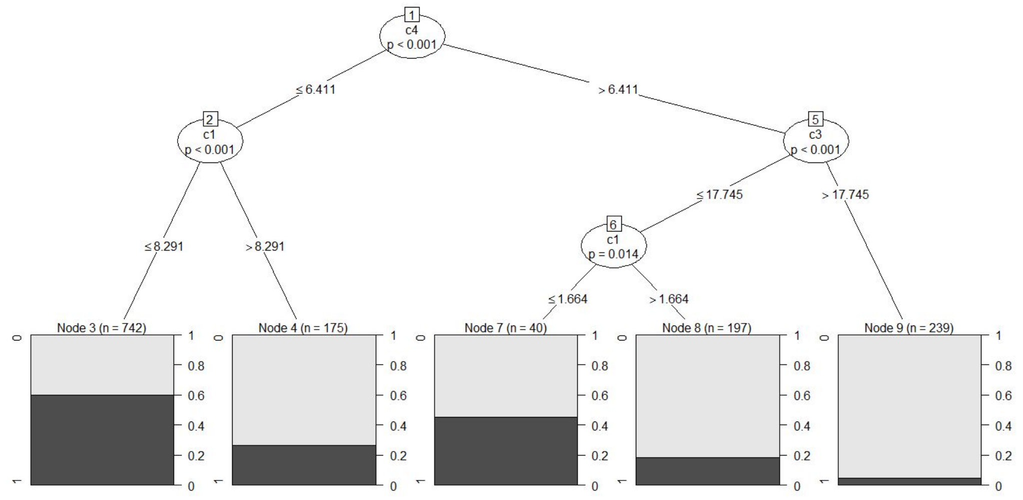

Figure 1 The above is a Decision Tree for the Polynomial Coefficients. The nodes that contributed most to classification error were Node 5 and Node 7.

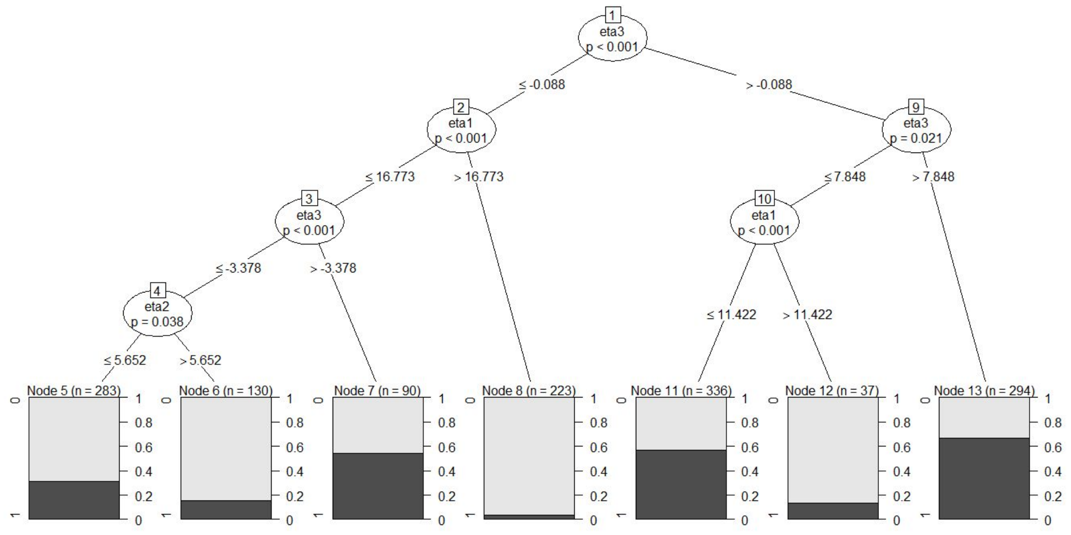

Figure 2 The above is a Decision Tree for the Eigendecomposition. The nodes that contributed most to classification error were Node 7, Node 11, and Node 13.

Comparison of Machine Learning Methods

The aforementioned machine learning methods of SVM, QDA, and Random Forests trained on the data's polynomial coefficients and eigenfunctions were compared for accuracy, sensitivity, and specificity.

Accuracy measures how much percent of the time the machine learning method correctly predicted whether or not the shot was made or missed.

Sensitivity is the measure of proportion of positives that are correctly identified. In the context of our problem, it measures how well the algorithm predicts made shots.

Specificity is the measure of proportion of negatives that are correctly identified, so it measures how well missed shots are made.

The values of accuracy, sensitivity, and specificity were calculated and presented in the table Figure 3. We fit 10 models in total to our data, using different permutations of Machine Learning algorithms and decomposition methods.

model	data	subjects	train	test	accuracy	sensitivity	specificity
QDA	Polynomial Coefficients	All shots	1000	393	0.6972	0.7197	0.6822
QDA	Polynomial Coefficients	Jump shots	400	251	0.7161	0.7777	0.6927
QDA	Eigenfunctions	All shots	1000	393	0.6946	0.6815	0.7033
QDA	Eigenfunctions	Jump shots	400	251	0.7625	0.6315	0.8086
SVM	Polynomial Coefficients	All shots	1000	393	0.6488	0.5325	0.7366
SVM	Polynomial Coefficients	Jump shots	400	251	0.8675	0.6779	0.9314
SVM	Eigenfunctions	All shots	1000	393	0.6870	0.6012	0.7478
SVM	Eigenfunctions	Jump shots	400	251	0.8990	0.7692	0.9397
Random Forest	Time Points	All Shots	1000	393	0.7531	0.7405	0.7617
Random Forest	Poly+Eigen	All Shots	1000	393	0.6972	0.6708	0.7148

Figure 3 The above table shows the Machine Learning model used for different features of our dataset.

Algorithm Accuracy

The Machine Learning Methods For “All Shots”, the most accurate classification ensemble was Quadratic Discriminant Analysis on the Polynomial Coefficients with 0.6792. The most accurate classification ensemble above is using Support Vector Machine with Eigenfunctions with a score of 0.8990.

Classifying Passes

In this second portion of our project, we wished to predict which player would receive the basketball next.

Algorithm

To classify passes, the `gradientAlgorithm()` function as documented in Appendix B was used. This function calculates the gradient of our data for each x-y coordinate in our dataset. The function then adds the gradient multiplied by a scalar to each x-y coordinate respectively. The nearest player to the ball according to a particular time interval is found and is predicted to be the person to receive the basketball next. When we initially examined the data, we noticed that passes are not guaranteed to be passed directly to the player, but rather where they were going. So we used the directionality to measure intended passes.

Algorithm Accuracy

Since we have to choose a scalar to weight our gradients in the algorithm, we have to find a way to fine-tune the weight parameter. To do this, we ran the algorithm iteratively over a single game. Our results are shown in Figure 4.

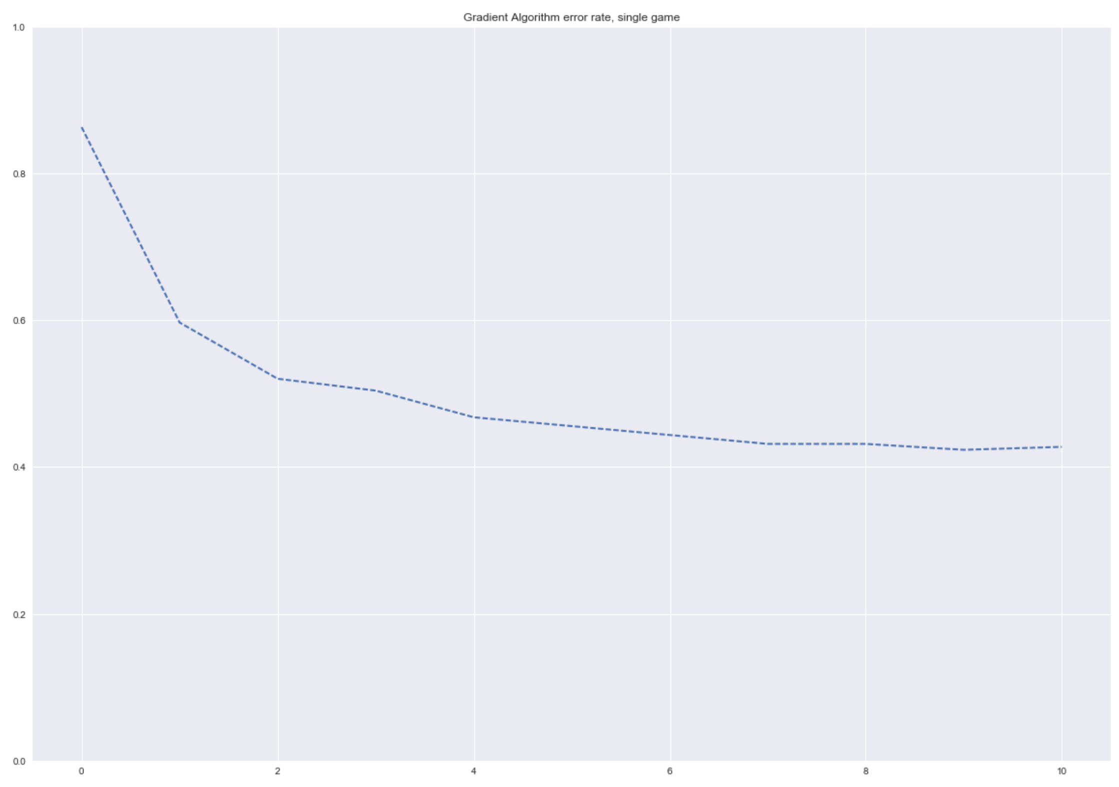

Figure 4 The plot above shows scalar weights on the x-axis and the corresponding error rates (of whether or not the pass went to the player we predicted or not) of the gradient algorithm on the y-axis.

With the initial point being **w=0**, we observe an error rate near 90%, but as we reach **w=6**, the results stabilize to an accuracy rate a bit above 40%. We decided to test the algorithm on a random sample of 30 games, iterating through the weights **w=6:10** and then averaging the error rates. Our results are presented in Figure 5.

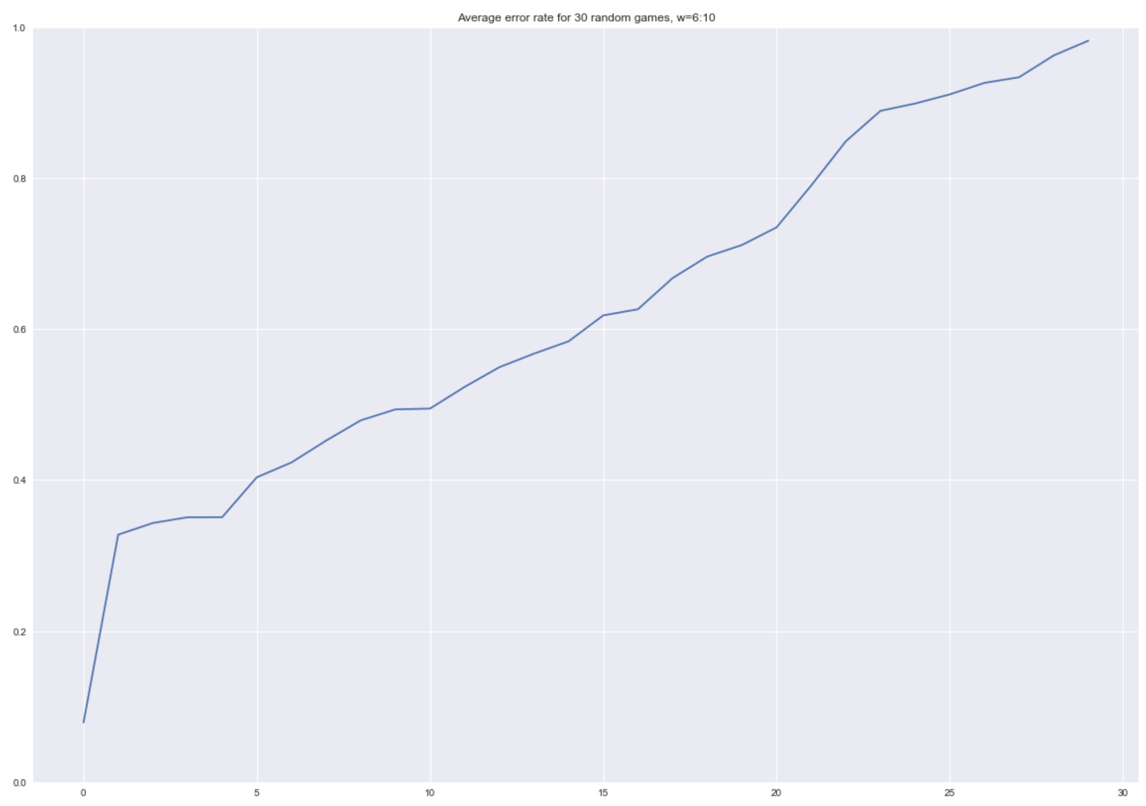

Figure 5 The plot shows 30 games on the x-axis and the corresponding average error rate is shown on the y-axis.

The first issue we noticed is there is a large variation in error rates across the games. 20 of the 30 games seem to have significantly larger error rates, which indicates that either the algorithm does not work on these games, or there are significant data issues with the games we were working with. Nonetheless, for well-behaved games, we achieve an error rate below 50%, which is non-trivial given that the algorithm is trying to predict between 9 different targets, which are the other basketball players.

Conclusion

During our 10-week project, we feel that our findings and algorithms are important initial steps to making a ball movement system. Our shot finding algorithm was quite accurate in finding shots when accounting for missing data and noise within the motion data. The classification methods used are steps in the right direction, since it is apparent that we can achieve high levels of accuracy when predicting misses or makes on jump shots. The next step is to account for noise within our data and find a better way to predict misses or makes on layups and other shot types. Our passing algorithm has room to improve. The gradient approach makes sense theoretically and should be able to be refined over time to achieve higher levels of accuracy. Overall, we feel that we made noticeable initial strides in creating a ball movement system that can be applied to the SportVu motion data.

Limitations

Over the course of the project, there were many types of data that we could not handle intelligently without the use of more advanced statistical methods. The types of plays we attempted to classify obviously did not cover the entire collection of possible basketball plays. Beyond these two limitations was the limitation of time in which we were unable to delve deeper into our dataset than we could have without the 10 week time constraint. Furthermore, it took us a lot of time to process and format the data into files that could be mined for patterns. Despite this, our findings showed the potential to be applied to current and future ball movement systems.

Future Work

Following Project Slam Dunc, there are many frontiers available for future work. Our findings have shown that there exist natural clustering between the different fitted model coefficients. However, we also observed a significant amount of noise between these clusters. Although we accounted for most of the noise, our shots data set still had a significant amount of it. Therefore, we urge future work to find methods to work through the noise to predict better results.

Second, we were unable to predict the outcome of a shot during the first 20% of the distance of a shot. We believe there to be values that can be used to classify a shot before it is near the hoop. Pairing an animation with the probability of the shot being made or missed as time went on would be of interest to both the expert and general basketball audience.

Last, incorporating more field values to predict whether shots were made or missed would be a canonical next step for basketball play classification. The variables we would most be interested in including in an updated version of the shot algorithm would be the player's launch angle, the player and ball's x-y position, the expected X and Y value of the ball near the rim, and the ball's velocity.

The ideal solution for a project of this caliber would be have annotated ball motion data specifying parabolas for shots, passes, dribbles, and other basketball plays. That way we could split the parabolas easily and run it through a recurrent neural network to get models with high levels of accuracy. However, this involves a large time commitment, access to game film to annotate the data, and access to GPUs to train the models.

Acknowledgements

We would like to acknowledge Professor Duncan Temple Lang and Nick Ulle for their support and guidance as we carried out this project. Thanks to Hans-Georg Mueller for writing the `fdapace` package.

Sources

[1] http://www.interbasket.net/news/20247/2016/11/most-popular-sport-in-the-world-by-country/

[2] http://variety.com/2017/tv/news/nba-regular-season-ratings-1202031083/

[3] https://www.stats.com/sportvu-basketball-media/

[4] https://www1.cfnc.org/Plan/For_A_Career/Career_Profile/Career_Profile.aspx?id=e8SgzgybeZFjv1aKkDSAPQXAP3DPAXXAP3DPAX

[5] https://www.mathworks.com/matlabcentral/fileexchange/62061-multi-class-svm

[6] https://onlinecourses.science.psu.edu/stat857/node/80

[7] https://en.wikipedia.org/wiki/Random_forest

[8] https://www.r-bloggers.com/why-do-decision-trees-work/

Appendix A: Reflection

Project Slam Dunc was definitely Patrick's mind child, as he'd been wishing to do this basketball project since August 2016. He collected the data used in this project back then. This project took a lot of time, collaboration, and communication amongst us three. Overall, the data savvy amongst team members was indubitable, and we all played to our strengths whether it be on-screen data manipulation or the communication of machine learning theory.

The data preprocessing step was challenging and took a long time to complete. Working with 50 gigabytes of data was new to us all, and using shell scripts to handle these data was also alien at first. Chirag was pivotal in making the preprocessing more efficient for the size of our data and our machines. Working on Project Slam Dunc, we have a better understanding on what reformatting "Big Data" into a convenient format can be like.

Another challenge was fitting models to the ball motion data. There were many models to choose from in order to produce fitted parameters. After digging through many Wikipedia pages and SciKit Learn documentation pages, we came across a lot more types of model fitting we never learned in class such as splines and Principle Component Analysis (PCA). It occurred to us that the data we were handling was not easily modeled in any particular way, and as far as Statistics goes, we have a lot more ground to cover and a lot more to learn.

Without a doubt, Project Slam Dunc has shown potential, and perhaps with some years of veteranship as Statisticians, we'll be able to understand our data on a level that it calls for. There were many limitations, but as stated in our conclusions, we have found some classifiers that work for a strong majority of the time, and that means for us that we have successfully completed our project.

Appendix B: Essential Code

Script 1: Shot Algorithm

```
def makeVoteSpace (motionXY, gameshotsXY):
    #initialize variables
    shots = []
    remove = []
    count = 0
    for i in gameshotsXY.as_matrix():
        #initialize variables
        balls = pd.DataFrame()
        players = pd.DataFrame()
        #get motion that is relevant to shot
        tempMotion = motionXY[(motionXY['quarter'] == i[3]) &
(motionXY['sec_in_quarter'] >= i[2]+0.01) &
                                (motionXY['sec_in_quarter'] <= i[2] + 4)]
        #try/except to remove shots that are missing from motion data
        try:
            #get playerID
            player_id = np.where(tempMotion.isin([i[4]]))[1][1]
            #get player and ball x/y coordinates
            players['x'] = tempMotion.iloc[:, player_id+1]
            players['y'] = tempMotion.iloc[:, player_id+2]
            balls['x'] = tempMotion['ball_x']
            balls['y'] = tempMotion['ball_y']
            #get estimated ball coordinates
            game = np.zeros(2)
            game[0] = i[0]
            game[1] = i[1]
            #figure out which side the shot is being taken
            if i[5] == 'Left':
                side = 5
            else:
                side = 89
            #this works, so I kept it that way, there are definiately ways to
make this more efficient but it works
            ballx = tempMotion['ball_x']
            bally = tempMotion['ball_y']
            #get difference in x/y coordinates
            gradientx = np.diff(ballx)
            gradienty = np.diff(bally)
            #get slope and intercept
            gradient_slope = [gradienty[j]/gradientx[j] for j in
range(len(gradientx))]
            gradient_intercept = [bally.iloc[j] -
gradient_slope[j]*ballx.iloc[j] for j in range(len(gradientx))]
            #predict y
            y = [gradient_slope[j]*side + gradient_intercept[j] for j in
range(len(gradientx))]
            #add 0 to get correct size of y
            y = [0] + y
            #make everything a matrix
            players = players.as_matrix()
            balls = balls.as_matrix()
            #get distances
            distance = [np.linalg.norm(players[j] - game) +
np.linalg.norm(balls[j] - game) + (y[j] - 25)**2 for j in
range(len(players))]
            #get indices
            ind = np.argpartition(distance, 5)[0:10]
            ind = [tempMotion.iloc[j].name for j in ind]
            #append indices to shots
            shots.append(ind)
        except (IndexError, ValueError):
            #if shot is not found append whatever
            shots.append([0,1,2,3,4,5,6,7,8,9])
            #add it to remove to remove in future
            remove.append(count)
        count = count + 1
    #delete shots from shots
    for i in sorted(remove, reverse=True):
        del shots[i]
    return (shots, remove)

def findShots (motion, shots):
    #get minima and initialize shot parabola
    minima = getRelMinima(motion)
    shot_parabola = []
    for i in shots:
        votespace = []
        #create votespace for shots
        for j in i:
            start = bisect_right(minima, j)-1
            votespace.append(start)
        #get start index by taking mode of votespace
        start = mode(votespace)[0][0]
        #try/except to make sure end and beginning indices are included
        try:
            end = minima[start+2]
        except IndexError:
            end = len(motion) - 1
        #make sure indices are within range
        if start-1 < 0:
            start = 0
        else:
            start = minima[start-1]
        shot_parabola.append([start, end])

    return shot_parabola
```

Script 2: Gradient Algorithm

This algorithm takes in the **motion** data, a 2d array of all XY coordinates in the game (**XY**), the list of player names and their id's (**player_names**), the set of passes from the run length data (**pass_table**), the set of index intervals **T** where the ball is being passed, and a scalar weight **w**.

```
def gradientAlgorithm(motion,XY,player_names,pass_table,T,w):
    #Evaluate the gradient of XY:
    XYgrad=np.gradient(XY,axis=0)
    #Weight the gradient values
    XYstar=XY+w*XYgrad
    #Find the distances to the ball
    d_star=np.linalg.norm(XYstar-XYstar[:,0,:].reshape(len(motion),1,2),axi
    s=2)
    #Find the closest object to the ball, based on gradient adjusted values
    d_min=d_star.argsort(axis=1)[:,1]-1
    min_df=pd.DataFrame(d_min,columns=["search"])
    #Create a dictionary to map numbers to column names and apply to the
    search.
    column_dict=dict(zip(np.arange(0,12),player_names.columns))
    min_df["search"]=min_df.search.map(column_dict)
    #Map the results from d_min to actual names
    weak_target=player_names.lookup(min_df.index,min_df.search.values)
```

```
#Predict the label with findTarget
pass_predictions=np.array([findTarget(weak_target[np.arange(t[0],t[1])]
)for t in T])
#Create a confusion matrix
class_table=pd.crosstab(pass_table.p2,pass_predictions)
#Generate an error rate and return it.
error_rate=1-np.sum(np.diag(class_table))/len(pass_table)
return(error_rate)
```

Appendix C: Additional Notes

Most useful data is in the datasets folder in GitHub. All game data can be made available upon request. Our GitHub repository contains some more visuals.