

An Analysis of NFL Teams Using MDS

TianYang Jin, Neeraj Suneel Joshi, TianChen Luo

March 9, 2016

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Background | 3 |
| 3 | The model itself and actual implementation | 4 |
| 3.1 | Goodness of Fit | 5 |
| 3.2 | Another approach to measure Goodness of Fit | 9 |
| 3.3 | Challenges during implementation | 9 |
| 3.4 | 1-D Plot? | 10 |
| 4 | Solution and Commentary on solution | 10 |
| 5 | Variations | 12 |
| 5.1 | Variation 1 | 12 |
| 5.2 | Variation 2 | 14 |
| 5.3 | Variation 3 | 15 |
| 6 | Conclusion | 18 |
| 7 | References/bibliography | 18 |
| 8 | Appendixes | 18 |

1 Introduction

In this day and age, measuring the success of a sport team is heavily dependent on statistics and data. Our team has chosen to analyze the differences and similarities across National Football League (NFL) teams during the years of 2015. American football is a team sport with a wide range of skilled positions and separate squads for different stages of the game. Specifically, a NFL team can have 11 players on the field at a given time, but requires three sets of players to remain competitive in the league: an Offensive unit, a Defensive unit and a Special teams unit. Because of the variability in the different positions and units, NFL teams can be highly skilled at different aspects of the game which lead to stronger statistics in a particular category describing the skill. In order to better visualize the different strengths and weaknesses of the NFL teams, we will use Multidimensional Scaling (MDS) in the hopes of creating insight on the success and failure of particular teams in the National Football League.

2 Background

From the very beginning, our group wanted to pursue a project which involved the application of Multidimensional Scaling due to our interest in the concept as well as the wide range of possible problems which involve comparing objects in a multidimensional sense. We decided to analyze sports teams because they rely so heavily on data and can be described using a multitude of statistics and metrics. Due to the high variability in the positions and responsibilities of NFL players described in the introduction, we decided that American football and the National Football League would provide the best platform to apply the technique of Multidimensional Scaling.

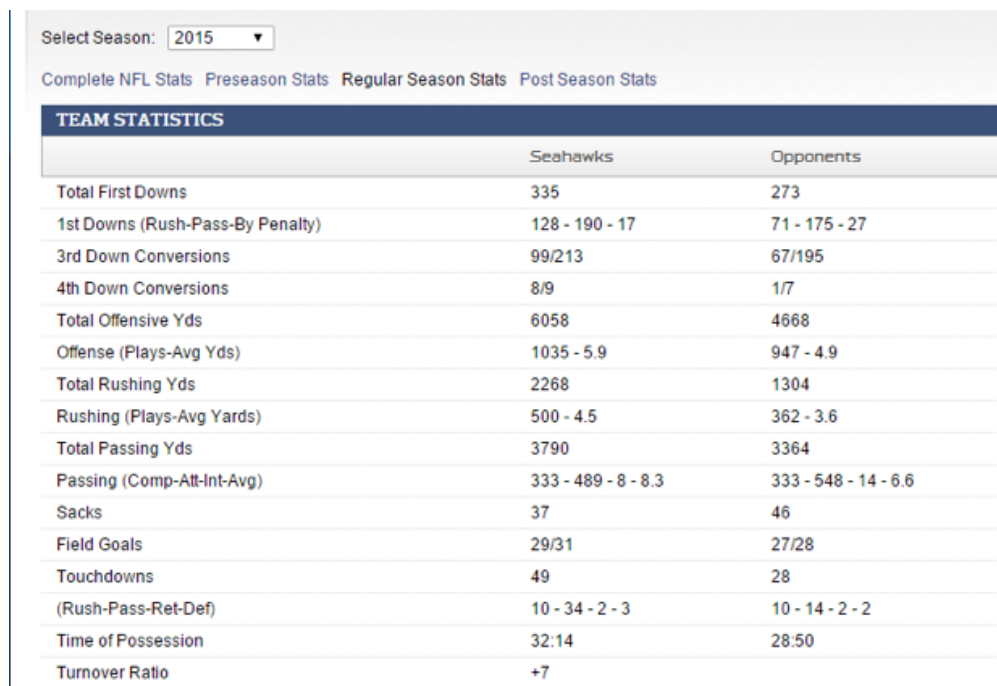
While the analysis of NFL teams is the constant subject of sports broadcasters, writers and commentators, the analysis of NFL teams in a multidimensional sense is not as prevalent as we thought it would be. A majority of NFL analysis is done with respect to individual players. This can be attributed to the popularity of Fantasy Football, a game in which football fans use their knowledge of the sport to compete against each other by managing and selecting NFL players, whose performance in real life contributes to the success of the fantasy team. The website, NFL.com, provides statistics on each NFL team and provides an analysis of the teams based on single statistics like yards per game, touchdowns, 1st downs, etc. While this is helpful for validating some of the results we obtained, it didn't provide the necessary multidimensional analysis that we were hoping to see. We could see the teams with the most number of touchdowns or the most yards per game, but no rankings were provided which visualized offensive performance or defensive performance as a whole. Sites like ESPN.com and other football central sites provided the same sort of single tiered analysis with respect to the offensive and defensive aspects of a team.

A very similar analysis was done on NFL teams during the 2006 season by Charles Kufs, author of *Stats with Cats*. In the blog he maintains, Kufs carries out an analysis of all 32 NFL teams with Factor Analysis and Multidimensional Scaling techniques. Factor analysis is a data reduction technique in which "the variation in a set of variables can be rearranged and attributed to new variables, called factors" (Kufs). Basically, factor analysis reduces the number of variables required in the analysis by comprising each factor with the weighted sum of the original variables. Kufs created an Offensive Factor and a Defensive Factor by weighting the original statistics of the NFL teams with respect to their importance to the two factors. He then used the technique of MDS to accurately create distances between teams to visualize them in a 2D plot. The axes of the corresponding plot measured the Offensive and Defensive factors of the NFL teams. While Kufs and our group are analyzing the same objects (NFL teams), we are solely relying on the technique of MDS to determine the distances between the NFL teams. Thus, different statistics will not be weighted higher or lower based on certain factors. They will, however, be normalized in order to create uniformity between statistics of different magnitudes. One of the outcomes of MDS is to determine what certain metrics can tell us about the distances between objects and how these objects will be aligned in a lower-dimensional plot. It may very well be that the axes correspond to the offensive and defensive strengths and weaknesses of a team, but this will rely on the set of statistics we choose to incorporate in the distance function.

3 The model itself and actual implementation

Multidimensional Scaling (MDS) aims to compress a high number of dimensions of data into a lower dimensional graph relaying important distance-like relationships between the data so that the similarities and dissimilarities among the objects in the data are easier to see. In our case, the objects in the data are the NFL teams and the distance between the objects is calculated using a distance function which takes the statistics we want to examine into consideration. To model the problem of analyzing various NFL teams through Multidimensional Scaling, we had to obtain the team statistics for each NFL team, normalize the data that we obtained, decide on which team statistics to use and finally create a distance matrix between the objects to input into the MDS function.

We scraped our data from the NFL.com website, which provides statistics for each team during the regular season as well as the post season for different years of play. We then created a python script which scraped the statistics for all 32 NFL teams given a year and time during the season (regular season, postseason) and finally placed all the data into a .csv file. A snapshot of data is shown in Figure 1 below, which details various statistics for the 2015 regular season for the Seattle Seahawks. Statistics for the Seattle Seahawks are provided as well as for the opponents they faced during the season.



| TEAM STATISTICS | | |
|----------------------------------|---------------------|----------------------|
| | Seahawks | Opponents |
| Total First Downs | 335 | 273 |
| 1st Downs (Rush-Pass-By Penalty) | 128 - 190 - 17 | 71 - 175 - 27 |
| 3rd Down Conversions | 99/213 | 67/195 |
| 4th Down Conversions | 8/9 | 1/7 |
| Total Offensive Yds | 6058 | 4668 |
| Offense (Plays-Avg Yds) | 1035 - 5.9 | 947 - 4.9 |
| Total Rushing Yds | 2268 | 1304 |
| Rushing (Plays-Avg Yards) | 500 - 4.5 | 362 - 3.6 |
| Total Passing Yds | 3790 | 3364 |
| Passing (Comp-Att-Int-Avg) | 333 - 489 - 8 - 8.3 | 333 - 548 - 14 - 6.6 |
| Sacks | 37 | 46 |
| Field Goals | 29/31 | 27/28 |
| Touchdowns | 49 | 28 |
| (Rush-Pass-Ret-Def) | 10 - 34 - 2 - 3 | 10 - 14 - 2 - 2 |
| Time of Possession | 32:14 | 28:50 |
| Turnover Ratio | +7 | |

Figure 1: Team Statistics for the Seattle Seahawks during their 2015 regular season

Since the data we collected for different metrics varies in magnitude and is not always in the same unit, it is important to normalize across each statistic. Since we have knowledge of the population parameters (μ , σ), we can use a standard score to measure the number of standard deviations a particular data point is above the corresponding mean. The standard score of a raw score x is provided in Figure 2. Once the standard scores of each data point are recorded, we can proceed to defining a cost function which will determine the distance between two teams.

$$z = \frac{x - \mu}{\sigma}$$

Figure 2: The standard score of a raw score x

Since we are looking at various strengths, weaknesses and skills of the teams, we want to examine different sets of statistics to discover insights specific to the various measures of the game (Offensive Ability, Defensive Ability, Both). Some statistics (Touchdowns, Offensive Yards, First Downs, Plays, and Time of Possession) describe how well the Offensive Unit does while other statistics (Fumbles, Sacks, Interceptions, Yards Allowed) depict the play of the Defensive Unit. When analyzing football teams, it is important to recognize strengths and weaknesses in both the Offensive and Defensive Units as they are both complimentary to the team's success or failure. By selecting sets of statistics which solely describe the Offensive squad or the Defensive squad, we are able to visualize teams in a more narrow scope, leading to more specific and detailed insights.

In order to implement Multidimensional scaling in Python, we will be using the module scikit learn's method, sklearn.manifold.MDS. This method requires a distance matrix describing the distance between each of the objects (teams) to fit the data properly. Because the sets of statistics are constantly changing with every scope of analysis, we decided that a distance function should take in the set of statistics as input in addition to the two teams we want to compute the distance between. The function computes the sum of the absolute differences of the standard scores for each of the statistics specified.

$$\sum_{j=1}^N abs(\frac{X_{1j} - \mu_j}{SD_j} - \frac{X_{2j} - \mu_j}{SD_j}),$$

where N = total number of statistics that we are considering

X_{ij} = statistic j for team i

μ_j = the mean of statistic j for all teams

SD_j = the standard deviation of statistic j for all teams

Iterating through each pair of teams, we created the distance matrix by way of the distance function implemented. In addition, we decided that it would be easier to visualize if the values were set to be between 0 and 1 so each value of the distance matrix was divided by the maximum value in the matrix. Once the matrix was defined, it was a matter of specifying the number of dimensions as well as the associating label with each of the data points. Since we had stored the names of the teams in the exact order that the distance matrix was ordered, we could simply use the zip function in Python to pair each team name with its corresponding data point. Using the matplotlib module in Python, it was straightforward to use the annotate method and specify the label, location of data point and some additional inputs for the aesthetics of the text label.

3.1 Goodness of Fit

Now that we have a good looking model, we still need to find a way to assess how well our model fits the data by using some kind of measure. We decided to calculate the Goodness of Fit (GOF) value of our model, because by definition, GOF is the difference between a particular distance and its corresponding pseudo-distance ($d_{ij} - d'_{ij}$). This serves as an index to how well the distance between i and j is in the solution configuration from the value required to preserve an ordinal relation with the data. If there is no inversion, then the difference will be zero. Alternatively, the difference can be viewed as the residual from monotone regression i.e. an index of the difference between the solution distance and (an ordinal re-scaling of) the data. (http://www.unesco.org/webworld/idams/advguide/Chapt8_1_1.htm)

The MDS package in python provides us with a built-in function for calculating the raw stress, which is also a measure of how the distances in the configuration fit the data. However, raw stress is not a good measure of fit at all because it is not normalized, i.e., identical configurations with different sizes will have a different value of stress. Hence, we need a normalized measure to make stress independent of size or scale of configuration and norming its value between 0 (worst fit) and 1 (perfect fit).

Hence, we decided to use a normalized measure to assess the fit of our model, namely **Stress Squared Deviation**, which is given by the following formula.

$$\text{Stress SQDEV} = [\sum (d_i - \hat{d}_{ij})^2 / \sum (d_{ij} - \bar{d})^2]^{\frac{1}{2}}$$

where \bar{d} = mean of all d'_{ij} s and $\sum (d_i - \hat{d}_{ij})^2$ is the raw stress.

We implemented a function to calculate the GOF of a given fit in python, and we ran the function on our 2-D model and it returned the GOF value 0.741.

But again, what does 0.741 mean? 0.741 is kind of close to 1, which is good, but we still need to find some way to determine the accuracy of this number. To do that, we conducted an experiment and generated 20 random matrices with the same size as the original distance matrix. We ran MDS on all these random distance matrices and computed the GOFs, to find that they were all between 0.347 and 0.396. After looking at these numbers, we have much stronger faith in our 2-D model and we concluded that our model fit pretty nicely.

Now that we have perfected our 2-D model and have a sense about how good it is, we went on and increased the dimension of our model. Theoretically, the larger the dimension, the better the fit because virtually the more dimensions we have, the more space we have to fit our data points to better satisfy their distance relationships. Since our distance matrix is 32 by 32, we increased the dimension from 1 to 32 and plotted the GOF vs. Dimension graph as shown below.

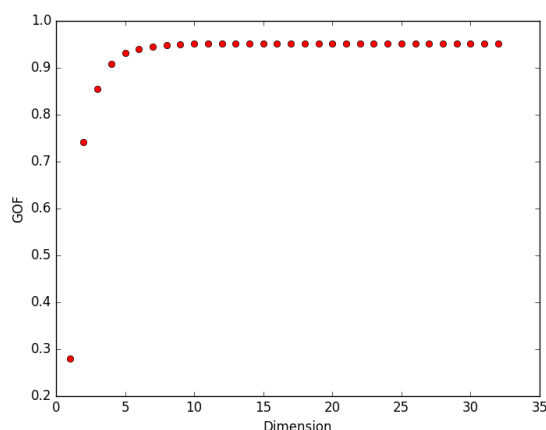


Figure 3: GOF vs. Dimension graph

From this graph, we can see that the GOF value stabilizes at about 0.922 after we increase the dimension up to and beyond 6. This figure meets our expectation regarding the increase in GOF with the increase in Dimension but yet, another unexpected problem arose. Theoretically, we should have a perfect fit for our model of dimension 32 because our original distance matrix has dimension 32. But that's not what happened, the GOF value for a dimension of 32 is also 0.922. We are not sure what caused this error, so we did another experiment and tried to figure out what went wrong.

For this experiment, we simply came up with 5 points on a 2-D plane, and manually computed the distances between every pair of points and constructed the distance matrix. And then we ran a 2-D MDS on this distance matrix and computed the GOF for it. We expected to see a perfect fit because we are mapping data points from a 2-D plane to another 2-D plane.

The 5 points that we used were (1, 1), (1, 2), (2, 1), (2, 2) and (3, 2). The actual graph consisting of the five points, the distance matrix, and the graph generated by MDS are shown below.

$$\begin{bmatrix} 0 & 1 & 1 & 1.4142 & 2.8284 \\ 1 & 0 & 1.4142 & 1 & 2.2361 \\ 1 & 1.4142 & 0 & 1 & 2.2361 \\ 1.4142 & 1 & 1 & 0 & 1.4142 \\ 2.8284 & 2.2361 & 2.2361 & 1.4142 & 0 \end{bmatrix}$$

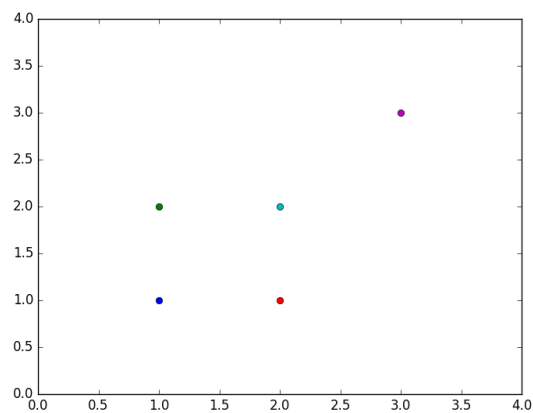


Figure 4: 5 points generated at exact location

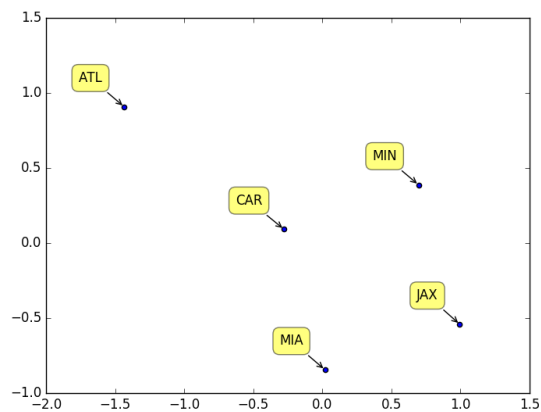


Figure 5: 5 points generated by MDS

Now we can virtually compare the two graphs and see if they are the same.
 If we see the graph generated by MDS at the following angle, we can conclude that this is a really close fit.

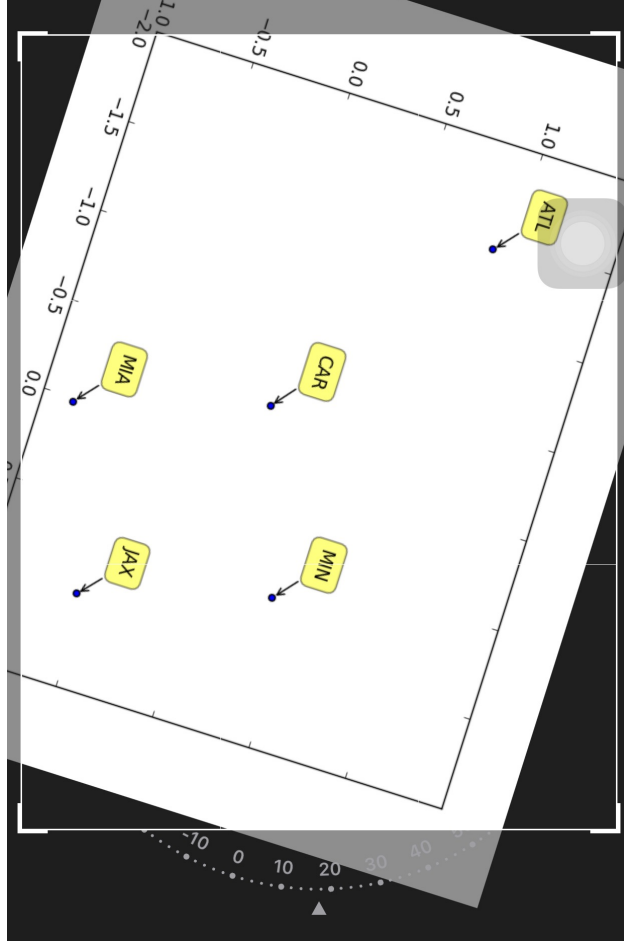


Figure 6: Rotated graph

However, the GOF value for this MDS model is not 1 either, it's 0.98. We then realized that the error is not caused by the MDS process or the way we are computing the GOF, but is caused by the rounding errors in the distance matrix. This makes sense because some of the entries in the distance matrix are irrational numbers like $\sqrt{2}$ or $\sqrt{5}$ which are truncated during the MDS process. In our actual model, mostly all the entries are truncated during the MDS process, and thus, it would make sense that we cannot get a perfect GOF value.

3.2 Another approach to measure Goodness of Fit

In this section, we will use a more straightforward and easier approach to measure the Goodness of Fit of our model. Generally, we will construct a new distance matrix, using the coordinates in our scaled matrix (the MDS-created distances matrix), and then compare this new distance matrix with the original distances matrix. Since we already have the original distance matrix, the problem now is constructing the new distance matrix.

First, we get all the coordinates of our resulting graph in 2 lists. The first list contains all of the X coordinates and the second list contains all of the Y coordinates. Thus, the first coordinate is $(X[0], Y[0])$ and the second is $(X[1], Y[1])$. Since this new matrix shares the same size as our original distance matrix, this new matrix is also 32 by 32. Suppose our new matrix is D , and d_{ij} is the element at the i^{th} row and j^{th} column. Then d_{ij} represents the distance from i^{th} coordinate to j^{th} coordinate in our coordinates lists. By implementing this method, we could fill in our new distance matrix with distances from every coordinate to every other coordinate. After constructing our new distance matrix, we used our original distance matrix as our X coordinates, and we used our new distance matrix as our Y coordinates. We could then construct a plot which contains 1024 (32 by 32) points, and this plot will tell us how the distances in the model compare with the original distances in our distance matrix. Original distances vs. distances in MDS-created model are shown below.

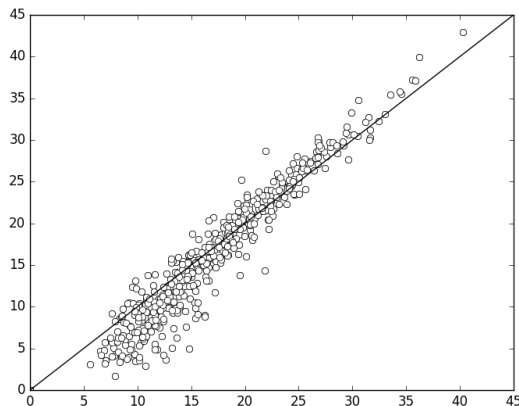


Figure 7: Original distances vs. distances in MDS-created model

In the graph above, we use a cross line to represent the perfect fit, which means the original distance matrix is the same as the coordinate distance matrix. From the graph, we could see that all points are close to the perfect fit line. Thus, we can conclude the Goodness of Fit in our model is fairly good.

3.3 Challenges during implementation

The biggest challenge that was encountered during the coding implementation was automating our data scraping and computation of the data matrix across different sets of statistics as well as different years or points in the season. Since our distance function is constantly changing with different metrics, it was challenging to create a simple way to input certain statistics and have the script locate those statistics in the .csv file. Our data scraping also had to be implemented in a way where the inputs included the year as well as the point of the season (regular season or post season) so that data across different periods of time could be easily obtained.

3.4 1-D Plot?

Originally we wanted to include a 1-D plot because we had thought that intuitively, it would make sense that teams could be ranked from the best performing to the worst performing. Ideally, we should get a straightforward 1-D plot showing how well the teams are doing. However, from figure 3, we can see that the GOF value for a 1-D fit is terrible. We tried to plot the 1-D graph using MDS, but the results were not impressive since most of the data points were clustered in the middle, and didn't provide much insight into how well they were doing relative to each other. Hence, we decided not to include the 1-D plot.

4 Solution and Commentary on solution

We decided to take the statistics we had for each team and separate them into solely offensive statistics, solely defensive statistics and also a combination of both. Out of the statistics we had, we decided on a certain subset of them to actually take into consideration. Some of the statistics did not make intuitive sense to include due to their relatively small importance into the offensive and defensive aspects of the game. For example, a statistic like 1st downs by penalty is totally dependent on the mistakes made by an opposing team and says little about the offensive strength of the team in question. The table below shows the statistics that were considered for both the offensive and defensive MDS implementations.

| Offensive Statistics | Defensive Statistics |
|-------------------------|---------------------------------|
| First Downs Passing | Sacks |
| First Downs Rushing | Defensive Touchdowns |
| Offensive Average Yards | Passing Interceptions |
| Offensive Plays | First Downs Passing Allowed |
| Passing Average Yards | First Downs Rushing Allowed |
| Passing Completions | Passing Completions Allowed |
| Passing Touchdowns | Passing Touchdowns Allowed |
| Rushing Average Yards | Rushing Average Yards Allowed |
| Rushing Plays | Rushing Plays Allowed |
| Rushing Touchdowns | Rushing Touchdowns Allowed |
| Total First Downs | Total First Downs Allowed |
| Total Offensive Yards | Total Offensive Yards Allowed |
| Total Passing Yards | Total Passing Yards Allowed |
| Total Rushing Yards | Total Rushing Yards Allowed |
| Touchdowns | Touchdowns Allowed |
| Time of Possession | Offensive Average Yards Allowed |
| | Offensive Plays Allowed |
| | Passing Average Yards Allowed |

For our first run-through of the MDS technique and visualization of the teams, we decided to focus solely on statistics involved with the offensive strength of a team. Running our distance function with the inputted statistics and creating the distance matrix, we were able to produce a 2D plot which showed further insight about the offensive prowess of the NFL teams during the 2015 regular season. The plot is shown in Figure 8 below for convenience.

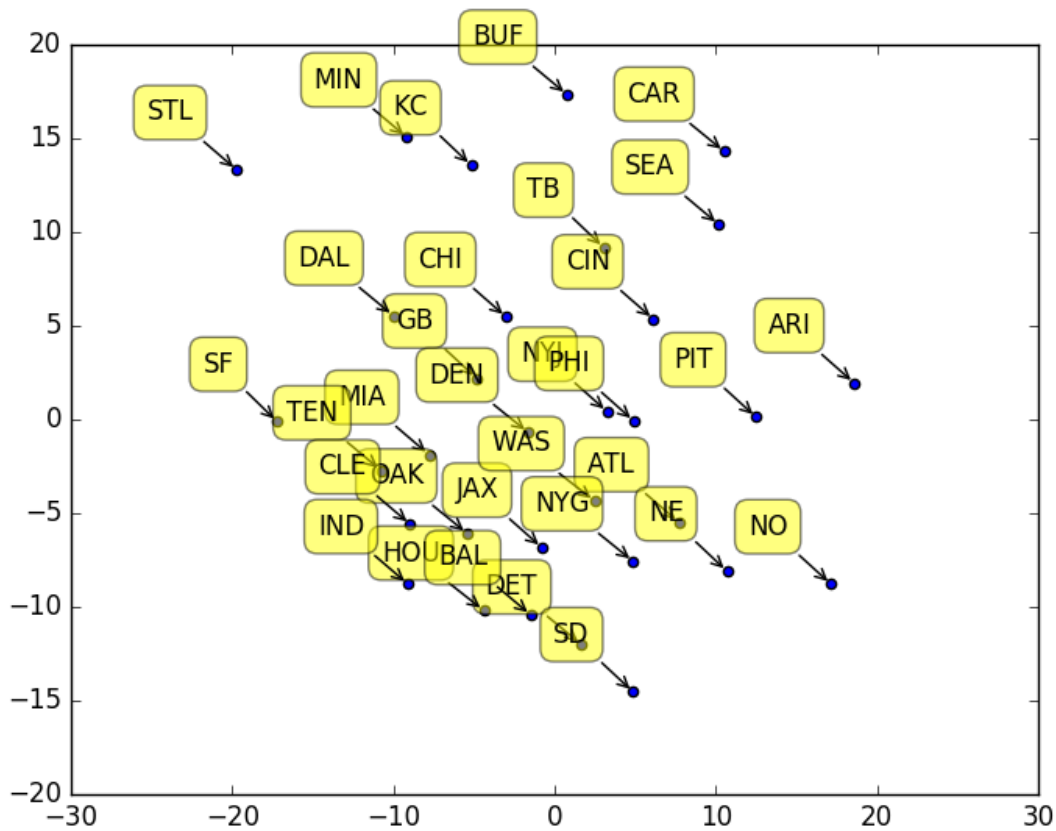


Figure 8: 2D plot showing the offensive performance of NFL teams during the 2015 regular season.

As seen in the above plot, the 32 NFL teams are plotted in a 2D space based on their respective distances away from each other. Looking at the teams and their relative performances in the 2015 season, it can be seen that the further to the right a team is placed, the better their rushing game. The rushing game is utilized by a football team when the quarterback runs the ball forward himself or when he hands the ball off to a running back who carries the ball forward for more yards. Looking at the team on the far right, it is seen that the Buffalo Bills have the best rushing game and this is indeed true. According to statistics found on NFL.com, the Buffalo Bills have the most rushing yards per game at 152 yards. The Bills also tied for the league high with 19 rushing touchdowns. In addition, “three players (LeSean McCoy, Tyrod Taylor, and Karlos Williams) all finished with over 500 rushing yards, and a fourth (Mike Gillislee) added another 267” (Galliford). When looking at teams who are placed to the far left, it can be seen that their rushing games were very poor during the 2015 season, with the San Diego Chargers (SD) and Detroit Lions (DET) having ranked last in rushing yards per game. On the other hand, looking at the y-axis, it can be seen that teams with high passing performances were located at the top of the graph while teams with a lackluster passing game were placed at the bottom. Teams like the Arizona Cardinals (ARI) and the New Orleans Saints (NO) lead the NFL with the most passing yards per game at 288.5 yards and 310.6 yards respectively. On the other hand, a team like the San Francisco 49ers struggled with their passing performance with the shaky performance and eventual injury of their star quarterback, Colin Kapernick. It is rare that teams in the NFL can rely solely on their rushing games or passing games and achieve success in the regular season. This is due to the fact that teams play against some defensive units that are generally better at stopping the running game while others have great secondary’s and can limit the success of the passing game. Thus, teams with high performance rushing and passing games will generally do the best in the league and win the most games during the season. Looking back to the 2D visualization, this corresponds to the teams located in the upper

right hand corner. The Carolina Panthers (CAR) were regarded to be the top performers in the NFL during the regular season as they only lost 1 game during their 16 game season. A large portion of their success comes from being offensively dominant, ranked first in number of points per game according to NFL.com. Stories like the ones above are littered throughout the 2D visualization and can accurately describe some of the successes and failures of teams through the season. The reason why MDS is implemented so well to this problem is that the statistics involved in computing the distances between the teams directly relate to the performances of the offensive and defensive units. By solely looking at offensive statistics, we are effectively narrowing our scope to see that the passing and rushing games are complimentary to an offensively dominant team.

5 Variations

5.1 Variation 1

For our first variation, we decided to examine the defensive statistics for each team to examine if the lower dimensional plot will give us more detail into what constitutes a defensively strong team vs a defensively weak team. With our distance function computing distances based solely on defensive statistics, we reconstructed our distance matrix and produced the 2D plot shown in Figure 9.

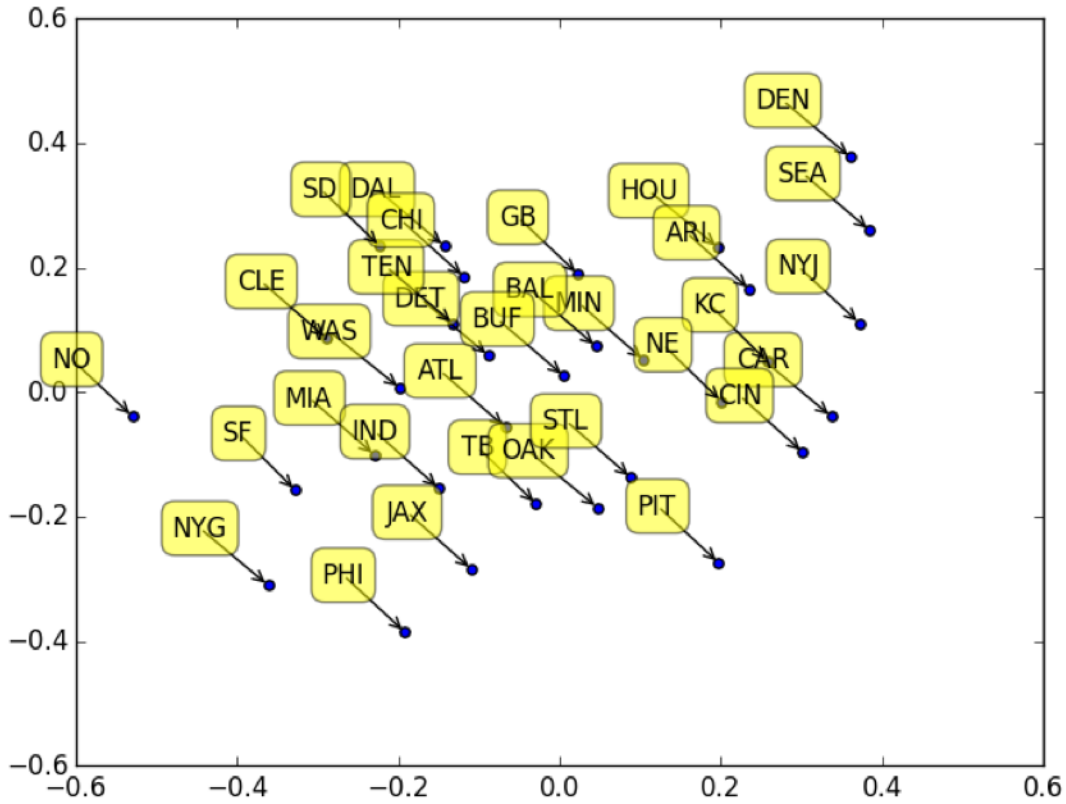


Figure 9: 2D plot showing the defensive performance of NFL teams during the 2015 regular season.

Looking at the teams and their distances apart, we noticed that teams along the y-axis showed significantly improved defensive performance against the passing game when moving towards the top. The Denver Broncos (DEN) were the furthest from any other team and the furthest to the top. When looking at statistics found on NFL.com, Denver lead the league in fewest passing yards allowed per game at 199.6 yards followed by the Seattle Seahawks (SEA) with 210.2 passing yards allowed per game. When looking at teams located towards the bottom, teams like the New Orleans Saints (NO) and New York Giants (SF) struggled throughout the whole season to cover wide receivers allowing some of the most passing yards per game at 284 yards and 298.9 yards per game respectively. When looking at the teams as they were horizontally placed, we noticed that the teams most skilled at preventing a good running game were placed at the right whereas teams who were susceptible to good running backs were placed at the left. The Seattle Seahawks (SEA), a team known for their dominant defensive line allowed the fewest number of rushing yards per game at 81.5 yards and allowed the fewest points per game at 17.3 points. This could be attributed to their strong defensive line made up of players like Michael Bennett and Cliff Avril as well as their star linebacker, Bruce Irvin. In a similar sense to the offensive visualization, the teams located on the upper right hand corner generally were strong in defending the passing and rushing game. Bear Bryant, longtime head coach of the University of Alabama football team, once said, “Offense sells tickets, but defense wins championships” (Wunderlich). This has become a sports cliché that sports announcers bring up often and has generally been shown as true with the recent super bowl winning teams. What’s interesting to note is that the Denver Broncos, who have been close to average in offensive performance, located in the center of the visualization, showed strong defensive power in the regular season and eventually went on to win the championship. We will take a closer look at the teams who made it to the post season and examine the corresponding offensive and defensive visualizations by way of MDS.

5.2 Variation 2

For our second variation, we decided to input both the offensive and defensive central statistics into the distance function and see how the corresponding 2D visualization will arrange the teams relative to their general performance in the regular season. Once the distance matrix was updated using the subset of relevant statistics, we used MDS to create the following plot.

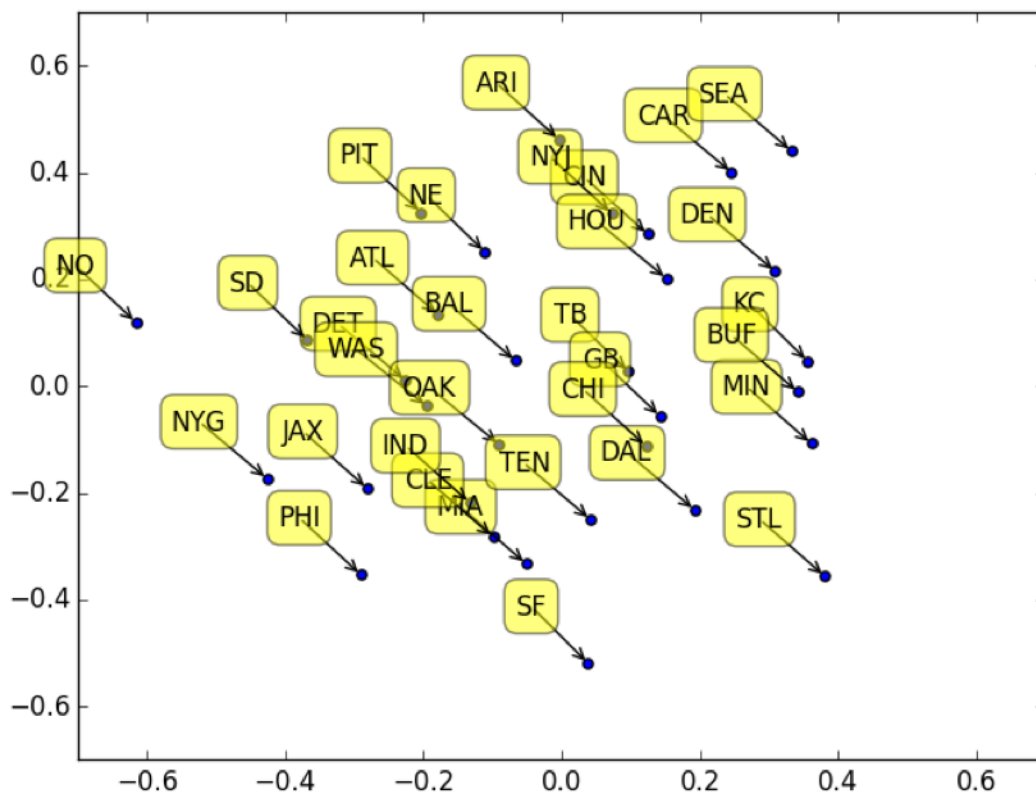


Figure 10: 2D plot showing the offensive and defensive performance of NFL teams during the 2015 regular season.

It is important to note that the general placement of the points in the horizontal and vertical position on the plot do not always correspond with the x and y-axis. The points may be off center and not placed in a way where the x-axis measures a certain trait in the data while the y-axis measures another feature of the data. In the above plot, it can be seen that teams that are placed closer to the upper left hand corner correlate with offensively strong performances while teams closer to upper right hand corner correspond with teams showcasing strong defensive talent. In this fashion, the strongest teams in both their offensive and defensive components will be located in the top middle portion of the visualization. An interesting takeaway from this visualization is that the teams located above the equation $y = 0.15$ all continued on to the playoffs, a measure of a teams success in the regular season. The fact that 9 out of the 12 teams are placed above a certain cut off line show the importance of having offensively and defensively strong units. The remaining three teams who made it to the playoffs are not far off from the cutoff line mentioned above. This could be due to the fact that their special teams unit was generally stronger than the rest of the teams and also because their offensive or defensive teams were very dominant. This is certainly the case with the Green Bay Packers (GB) whose elite quarterback, Aaron Rodgers carried them through close games down the stretch.

5.3 Variation 3

We decided that it would be equally insightful to examine the teams who made it to the playoffs during the 2015 - 2016 season to see how offensive and defensive strength played into winning and losing games of high importance. One major change we had to make in our algorithm was to account for the fact that certain teams played more games in the playoffs than others. This is because the NFL follows a 1 game elimination structure where a team cannot continue on if they lose. Thus teams who make it to the Super Bowl have to play 3 games, while teams who are eliminated in the first round only play 1 game. Since our statistics were cumulative through all games in the post season, we had to divide the statistics by the number of games the specific team played. We first began by examining the offensive statistics for the twelve teams who made it to the post season. Figure 11 shows the visualization that ensued after consolidating the data and calculating the distance matrix for all twelve teams.

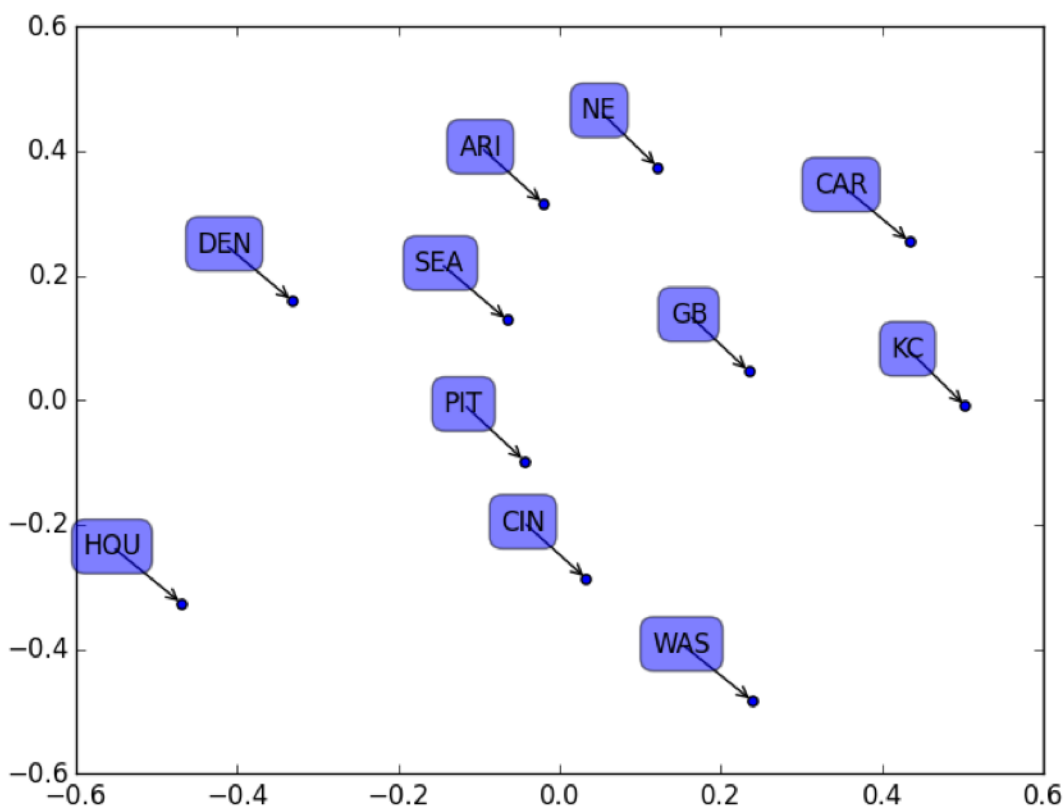


Figure 11: 2D plot showing the offensive performance of NFL teams during the 2015 post season.

Similar to the logic following the offensive visualization during the regular season, the teams towards the right show dominance in their rushing game while teams located at the top of the graph show strong passing performances. The strong passing performance from the New England Patriots (NE) was expected as their quarterback, Tom Brady “holds the NFL record of 31 for most number of playoff games starting” (Chase). Looking at solely defensive metrics of performance, we produced the following visualization to see how teams did against the passing and rushing game during the post season.

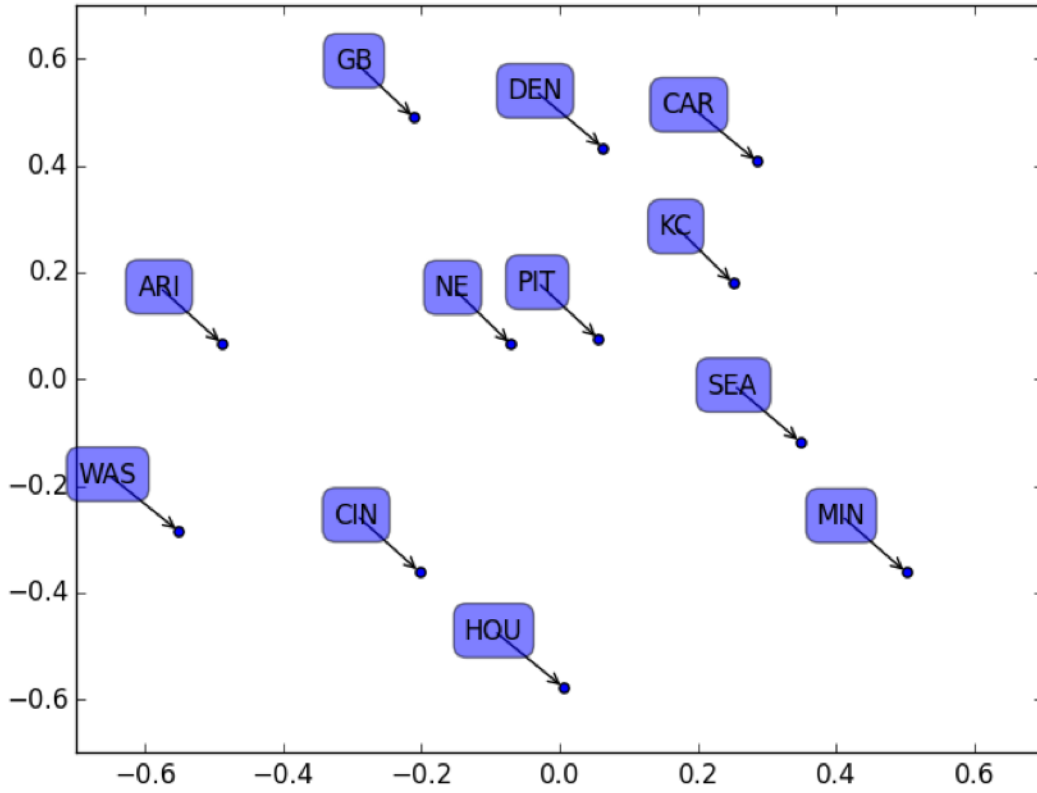


Figure 12: 2D plot showing the defensive performance of NFL teams during the 2015 post season.

Similar to the logic we followed in the defensive analysis of the regular season, the teams which were located towards the right in the 2D plot signified teams with high defensive ability against the passing game while teams located at the top portion of the graph correlated with teams who were adept at countering the rushing game. The outlier corresponding with the Minnesota Vikings (MIN) is due to the fact that they were eliminated in 1 game and held the Seahawks to only 10 points and 129 passing yards according to ESPN. Looking at teams with strong defensive capabilities in blocking both the rushing and passing sense, the Carolina Panthers (CAR) and Denver Broncos (DEN) dominated in both aspects and were located in the upper right hand side of the plot. This supports Bear Bryant and his statement suggesting that defenses do win championships. Finally, we decided to look at the offensive and defensive statistics relating to teams who made it to the postseason. The plot produced is given in Figure 13 below.

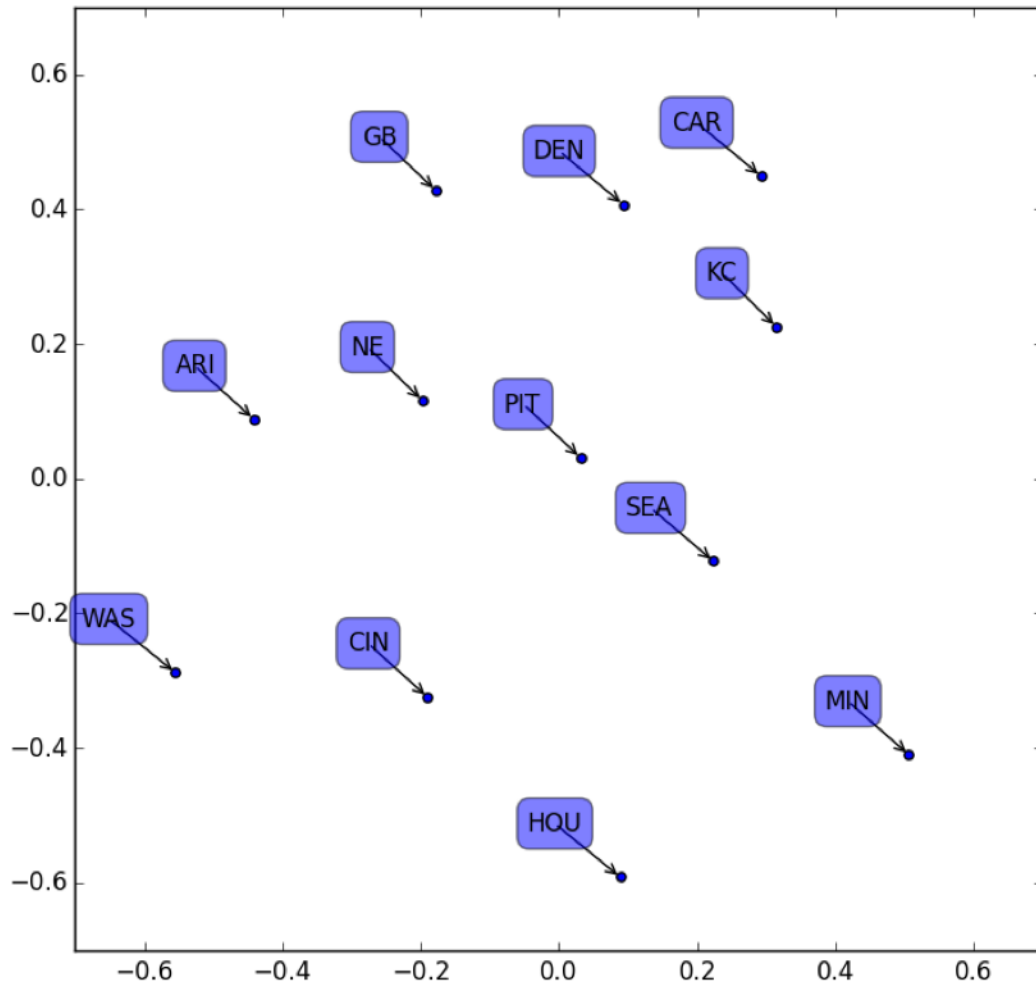


Figure 13: 2D plot showing the offensive and defensive performance of NFL teams during the 2015 regular season.

Looking at both the offensive and defensive statistics and applying MDS, we can see that the teams who show significantly better performance in offense are seen at the very right of the graph while teams who show exemplary performance in defensive capabilities are located at the northern portion of the plot. Looking at the teams who played well offensively and defensively, we can see the two teams who participated in the Super Bowl (Denver Broncos (DEN) and Carolina Panthers (CAR)) as well as the Kansas City Chiefs (KC) were located in the top right of the graph. According to the NFL box score, the Chiefs did very well in the playoffs but only lost by 7 points to the New England Patriots in the AFC Divisional round. This explains their outstanding performance but unfortunate elimination in the post-season. Another insight into this plot is the fact that statistically, the Carolina Panthers were the dominant team throughout the entire season as well as the playoffs in both their offensive and defensive abilities. However, when it came down to the last game, the Denver Broncos were able to contain Carolina's young and talented quarterback, Cam Newton, and stop him from creating plays down the field. This plays into the fact that Denver's team is led by Peyton Manning, a veteran quarterback, whose experiences and strong defensive unit gave him the upper

hand in the final game of the season. Insights like these can explain why certain outliers exist and why some successes and defeats are beyond the scope of just the statistics.

6 Conclusion

When we first began to examine the different statistics recorded for each football team in the NFL, all the numbers and different metrics were difficult to comprehend when considering the relative strength and weakness of a team. The lower dimensional visualizations made possible with multidimensional scaling helped consolidate the statistics into categories relevant to the success of an American football team. For example, after considering the offensive statistics, we witnessed that rushing and passing performances are necessary for a team to score points and be versatile against different defensive units in the league. Similarly defensive units need to be good at stopping the rushing and passing games of their opposing offensive units. When combining all of the statistics and visualizing the distances between the teams, it is seen that the strongest and best performing teams are located at the top ends of both the offensive and defensive spectrum. Our results can be useful in predicting the success of a team in the playoffs given their statistics during the regular season. However, while we can predict which teams will generally make it to the later rounds of the playoffs, factors like experience and stress are outside the boundary of statistics as seen in the 2016 Super Bowl with the statistically dominant Carolina Panthers losing to the more experienced Denver Broncos team.

7 References/bibliography

Brian Galliford. *Buffalo Bills offensive line gelled much faster than expected in 2015*. URL: <http://www.buffalorumblings.com/bills-news-notes/2016/1/8/10735520/buffalo-bills-offensive-line-2015-greg-roman-n>

Charles Kufs. *Statistics: a Remedy for Football Withdrawal*. URL: <https://statswithcats.wordpress.com/2011/02/27/statistics-a-remedy-for-football-withdrawal>.

David Wunderlich. *Was Bear Bryant Right About Offense Selling Tickets?* URL: <http://www.teamspeedkills.com/2011/7/28/2300735/offense-sells-tickets-defense-wins-championships>.

NFL. *Team Statistics*. URL: <http://www.nfl.com/stats/team>.

NFL. *PLAYOFFS*. URL: <http://www.nflplayoffsbracket.com/2016/index.php>.

8 Appendixes

Here are all the codes that we used to scrape the data and doing the MDS.

```
# we used Python 2.7.9 and the calculations for the MDS visualizations took  
    around 10 seconds for  
# each set of statistics
```

```
# this script takes in statistics involved with NFL teams and returns a 2D  
    plot of  
# the 32 teams in the NFL
```

```
import csv  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import manifold # package used for MDS implementation  
import math
```

```
# configuration to print entire np arrays
```

```

np.set_printoptions(threshold=np.nan)

# reading in .csv data
reader = csv.reader(open("nfl_team_stats2015.csv", "r"), delimiter=';')
readero = csv.reader(open("nfl_team_stats2015o.csv", "r"), delimiter=';')
data = list(reader)
datao = list(readero)

dists = []
teams = []

#
#####

column_letters = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
                  'M', 'N', 'O',
                  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
row_stats = ['Team', '1st_Downs_By_Penalty', '1st_Downs_Pass', '1st_Downs_Rush',
             '3rd_Down_Conversions', '4th_Down_Conversions', 'Defensive_Touchdowns',
             'Field_Goals', 'Offense_Avg_Yds', 'Offense_Plays', 'Passing_Attempmts',
             'Passing_Average', 'Passing_Completions',
             'Passing_Interceptions', 'Passing_Touchdowns', 'Return_Touchdowns',
             'Rushing_Avg_Yds', 'Rushing_Plays', 'Rushing_Touchdowns',
             'Sacks', 'Time_of_Possession', 'Total_First_Downs', 'Total_Offensive_Yds',
             'Total_Passing_Yds', 'Total_Rushing_Yds', 'Touchdowns',
             'Turnover_Ratio']

nfl_teams = ['JAX', 'MIN', 'MIA', 'CAR', 'ATL', 'TB', 'DET', 'CIN', 'NYJ',
             'HOU', 'GB', 'DEN', 'BAL', 'WAS', 'NYG', 'OAK', 'KC', 'PHI', 'TEN', 'NO',
             'DAL', 'PIT', 'NE', 'CLE', 'STL', 'SEA', 'CHI', 'IND', 'ARI', 'BUF', 'SF',
             'SD', 'AVERAGE', 'SD']

# this method takes as input a list of different statistics and returns the
# indices
# of those statistics as they are found in the .csv data file
def stats_to_columns(stats):
    cols = []
    for stat in stats:
        index = row_stats.index(stat)
        cols.append(index)
    return cols

# these statistics are the ones involved in offensive performance
stats_of_interest = ['1st_Downs_Pass', '1st_Downs_Rush', 'Offense_Avg_Yds',
                    'Offense_Plays', 'Passing_Average', 'Passing_Completions',
                    'Passing_Touchdowns', 'Rushing_Avg_Yds', 'Rushing_Plays', 'Rushing_Touchdowns',
                    'Total_First_Downs',
                    'Total_Offensive_Yds', 'Total_Passing_Yds', 'Total_Rushing_Yds',
                    'Touchdowns', 'Time_of_Possession']

# stats_of_interest = ['Sacks', 'Defensive Touchdowns', 'Passing Interceptions']
# stats_of_interest_defense = []

```

```

# these statistics are the ones involved in defensive performance
stats_of_interest_defense = ['1st_Downs_Pass', '1st_Downs_Rush', 'Offense_Avg_Yds', 'Offense_Plays', 'Passing_Average', 'Passing_Completions', 'Passing_Touchdowns', 'Rushing_Avg_Yds', 'Rushing_Plays', 'Rushing_Touchdowns', 'Total_First_Downs', 'Total_Offensive_Yds', 'Total_Passing_Yds', 'Total_Rushing_Yds', 'Touchdowns']

# this function calculates the distance between two teams given a certain list of offensive and defensive
# statistics
def cost(team1, team2, stats_interested, stats_interested_defense):
    cost_value = 0
    cols = stats_to_columns(stats_interested)
    cols_def = stats_to_columns(stats_interested_defense)
    for i in range(0, len(cols)):
        cost_value = cost_value + math.fabs(((float(data[team1][0].split(',')[cols[i]]) - float(data[33][0].split(',')[cols[i]]))/float(data[34][0].split(',')[cols[i]])) - ((float(data[team2][0].split(',')[cols[i]]) - float(data[33][0].split(',')[cols[i]]))/float(data[34][0].split(',')[cols[i]])))
    for i in range(0, len(cols_def)):
        cost_value = cost_value + math.fabs(((float(dataao[team1][0].split(',')[cols_def[i]]) - float(dataao[33][0].split(',')[cols_def[i]]))/float(dataao[34][0].split(',')[cols_def[i]])) - ((float(dataao[team2][0].split(',')[cols_def[i]]) - float(dataao[33][0].split(',')[cols_def[i]]))/float(dataao[34][0].split(',')[cols_def[i]])))
    return cost_value

# this portion of code uses the cost function defined above to calculate the distances
# between each pair of teams in the 32 team NFL league
for i in range(1, len(data) - 2):
    dist_team = []
    teams.append(data[i][0].split(',')[0])
    for j in range(1, len(data) - 2):
        dist_team.append(cost(j, i, stats_of_interest, stats_of_interest_defense))
    dists.append(dist_team)

# Runs MDS on a provided distance matrix,
# returns a result containing the coordinates for the MDS graph
def scaling(adist, dim):
    mds = manifold.MDS(n_components=dim, dissimilarity="precomputed", random_state=4)
    results = mds.fit(adist)
    return adist, results

# Run MDS for n times on n random generated distance matrices,
# return a list of the GOF values from all the MDS fits.
def run_random_experiment(n):
    rand_result = []

```

```

    for i in range(0, n):
        adist = random_symmetric_matrix(32)
        adist, results = scaling(adist, 2)
        s = results.stress_
        rand_result.append(get_gof(s, adist))
    return rand_result

# Plot a gof-vs-dimension graph,
# dimension ranges from 1 to the dimension of the original matrix (32 in this
# case).
def gof_vs_dim(adist, max_dim):
    X = range(1, max_dim+1)
    Y = []
    for dim in range(1, max_dim+1):
        adist, results = scaling(adist, dim)
        s = results.stress_
        Y.append(get_gof(s, adist))

    plt.plot(X, Y, 'ro')
    plt.xlabel('Dimension')
    plt.ylabel('GOF')
    plt.show()

# Plot the MDS graph using the coordinates stored in the result returned from
# MDS process
def MDS_graph(coords):
    plt.subplots_adjust(bottom = 0.1)
    plt.scatter(
        coords[:, 0], coords[:, 1], marker = 'o'
    )
    for label, x, y in zip(teams, coords[:, 0], coords[:, 1]):
        plt.annotate(
            label,
            xy = (x, y), xytext = (-20, 20),
            textcoords = 'offset_points', ha = 'right', va = 'bottom',
            bbox = dict(boxstyle = 'round,pad=0.5', fc = 'yellow', alpha =
                0.5),
            arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,radius=0
                '))

    plt.show()

# Reconstruct a distance matrix using the coordinates returned from the MDS
# process
def reconstruct_distance_matrix(X, Y):
    D_2 = np.zeros((32, 32))
    for i in range(0, 32):
        this_pt = (X[i], Y[i])
        for j in range(0, 32):
            curr_pt = (X[j], Y[j])
            D_2[i][j] = pt_dist(this_pt, curr_pt)
    return D_2

# Return the distance of two points on a plane

```

```

def pt_dist(P1, P2):
    return math.sqrt(math.pow(P2[0] - P1[0], 2) + math.pow(P2[1] - P1[1], 2))

# Plot the difference between the original distance matrix and the
reconstructed matrix after MDS
def D1_vs_D2(coords):
    X = coords[:, 0]
    Y = coords[:, 1]
    D_2 = reconstruct_distance_matrix(X, Y)

    plt.plot(adist, D_2, 'wo', range(0, 46), range(0, 46), 'k')
    plt.xlabel('Original_Distance_Matrix')
    plt.ylabel('Distance_Matrix_after_MDS')
    plt.show()

# Returns the max, min, and mean from n number of random matrix experiment
def assess_random_experiment(list):
    max = 0
    min = 1
    sum = 0
    for gof in list:
        if gof > max:
            max = gof
        if gof < min:
            min = gof
        sum += gof
    mean = sum / len(list)
    return max, min, mean

adist, results = scaling(adist, 2)
coords = results.embedding_
s = results.stress_
# print get_gof(s, adist)
# gof-vs-dim(adist, 32)
# D1-vs-D2(coords)
MDS_graph(coords)

# This script scrapes data from NFL.com for each of the 32 teams in the NFL
and writes the data out to
# two .csv files.
from bs4 import BeautifulSoup
import urllib2
import time
import csv

# dictionary linking each team in the NFL with its 3-letter acronym.
team_names = {'ARI': 'Arizona_Cardinals',
               'ATL': 'Atlanta_Falcons',
               'BAL': 'Baltimore_Ravens',
               'BUF': 'Buffalo_Bills',
               'CAR': 'Carolina_Panthers',
               'CHI': 'Chicago_Bears',
               'CIN': 'Cincinnati_Bengals',
               'CLE': 'Cleveland_Browns',

```

```

'DAL': 'Dallas_Cowboys',
'DEN': 'Denver_Broncos',
'DET': 'Detroit_Lions',
'GB': 'Green_Bay_Packers',
'HOU': 'Houston_Texans',
'IND': 'Indianapolis_Colts',
'JAX': 'Jacksonville_Jaguars',
'KC': 'Kansas_City_Chiefs',
'MIA': 'Miami_Dolphins',
'MIN': 'Minnesota_Vikings',
'NE': 'New_England_Patriots',
'NO': 'New_Orleans_Saints',
'NYG': 'New_York_Giants',
'NYJ': 'New_York_Jets',
'OAK': 'Oakland_Raiders',
'PHI': 'Philadelphia_Eagles',
'PIT': 'Pittsburgh_Steelers',
'SD': 'San_Diego_Chargers',
'SEA': 'Seattle_Seahawks',
'SF': 'San_Francisco_49ers',
'STL': 'Saint_Louis_Rams',
'TB': 'Tampa_Bay_Buccaneers',
'TEN': 'Tennessee_Titans',
'WAS': 'Washington_Redskins'}

```

lists and dictionaries associated with the data storage

```

all_team_stats = {}
all_team_stats_list = []
all_team_statso = {}
all_team_statso_list = []

```

iterating through each NFL team

```

for i in range(0, len(team_names)):
    url = 'http://www.nfl.com/teams/statistics?season=2015&team=' + team_names
        .keys()[i] + '&seasonType=REG'
    page = urllib2.urlopen(url)
    soup = BeautifulSoup(page.read(), 'html.parser') # package used to read
        HTML content of website

    team_stats = {}
    team_stats_list = []
    table = soup.find("table", {"class": 'data-table1_team-stats'})
    rows = table.find_all("tr")
    for j in range(2, 18):
        row_sections = rows[j].find_all("td")
        if j == 3:
            team_stats['1st_Downs_Rush'] = row_sections[1].get_text().split("-")
                [0].strip()
            team_stats['1st_Downs_Pass'] = row_sections[1].get_text().split("-")
                [1].strip()
            team_stats['1st_Downs_By_Penalty'] = row_sections[1].get_text().
                split("-")[2].strip()
        elif j == 7:

```

```

        team_stats['Offense_Plays'] = row_sections[1].get_text().split("-")
        [0].strip()
        team_stats['Offense_Avg_Yds'] = row_sections[1].get_text().split("-")
        [1].strip()
    elif j == 9:
        team_stats['Rushing_Plays'] = row_sections[1].get_text().split("-")
        [0].strip()
        team_stats['Rushing_Avg_Yds'] = row_sections[1].get_text().split("-")
        [1].strip()
    elif j == 11:
        team_stats['Passing_Completions'] = row_sections[1].get_text().
            split("-")[0].strip()
        team_stats['Passing_Attempts'] = row_sections[1].get_text().split(
            "-")[1].strip()
        team_stats['Passing_Interceptions'] = row_sections[1].get_text().
            split("-")[2].strip()
        team_stats['Passing_Average'] = row_sections[1].get_text().split("-")
        [3].strip()
    elif j == 15:
        team_stats['Rushing_Touchdowns'] = row_sections[1].get_text().
            split("-")[0].strip()
        team_stats['Passing_Touchdowns'] = row_sections[1].get_text().
            split("-")[1].strip()
        team_stats['Return_Touchdowns'] = row_sections[1].get_text().split(
            "-")[2].strip()
        team_stats['Defensive_Touchdowns'] = row_sections[1].get_text().
            split("-")[3].strip()
    elif j == 16:
        team_stats['Time_of_Possession'] = row_sections[1].get_text().
            split(':')[0] + '.' + str(float(row_sections[1].get_text().
            split(':')[1])/60.0).split('.')[1]
    else:
        team_stats[row_sections[0].get_text()] = row_sections[1].get_text
        ()

# storing the scraped data into the associated lists and dictionaries
all_team_stats[team_names.keys()[i]] = team_stats
time.sleep(1)
team_stats_list.append(team_names.keys()[i])
for i in range(0, len(team_stats)):
    team_stats_list.append(team_stats[sorted(team_stats)[i]])
all_team_stats_list.append(team_stats_list)

for i in range(0, len(team_names)):
    url = 'http://www.nfl.com/teams/statistics?season=2015&team=' + team_names
        .keys()[i] + '&seasonType=REG'
    page = urllib2.urlopen(url)
    soup = BeautifulSoup(page.read(), 'html.parser')
    # sum_touchdowns = 0
    # for key, value in all_team_stats.items():
    #     sum_touchdowns += int(value['Touchdowns'])
    # print sum_touchdowns
    team_statso = {}
    team_statso_list = []

```



```

table = soup.find("table", {"class": 'data-table1_team-stats'})
rows = table.find_all("tr")
for j in range(2, 18):
    row_sections = rows[j].find_all("td")
    if j == 3:
        team_statso['1st_Downs_Rush'] = row_sections[2].get_text().split("-")
        team_statso['1st_Downs_Pass'] = row_sections[2].get_text().split("-")
        team_statso['1st_Downs_By_Penalty'] = row_sections[2].get_text().split("-")
    elif j == 7:
        team_statso['Offense_Plays'] = row_sections[2].get_text().split("-")
        team_statso['Offense_Avg_Yds'] = row_sections[2].get_text().split("-")
    elif j == 9:
        team_statso['Rushing_Plays'] = row_sections[2].get_text().split("-")
        team_statso['Rushing_Avg_Yds'] = row_sections[2].get_text().split("-")
    elif j == 11:
        team_statso['Passing_Completions'] = row_sections[2].get_text().split("-")
        team_statso['Passing_Attempts'] = row_sections[2].get_text().split("-")
        team_statso['Passing_Interceptions'] = row_sections[2].get_text().split("-")
        team_statso['Passing_Average'] = row_sections[2].get_text().split("-")
    elif j == 15:
        team_statso['Rushing_Touchdowns'] = row_sections[2].get_text().split("-")
        team_statso['Passing_Touchdowns'] = row_sections[2].get_text().split("-")
        team_statso['Return_Touchdowns'] = row_sections[2].get_text().split("-")
        team_statso['Defensive_Touchdowns'] = row_sections[2].get_text().split("-")
    elif j == 16:
        team_statso['Time_of_Possession'] = row_sections[2].get_text().split(':')[0] + '.' + str(float(row_sections[2].get_text().split(':')[1]) / 60.0).split('.')[1])
    else:
        try:
            team_statso[row_sections[0].get_text()] = row_sections[2].get_text()
        except:
            team_statso[row_sections[0].get_text()] = 'NONE'
# storing the scraped data into the associated lists and dictionaries
all_team_statso[team_names.keys()[i]] = team_statso
time.sleep(1)
team_statso_list.append(team_names.keys()[i])
for i in range(0, len(team_statso)):

```

```

        team_statso_list.append(team_statso[sorted(team_statso)[i]])
    all_team_statso_list.append(team_statso_list)

# writing the data obtained to a .csv file
with open('nfl_team_stats2015.csv', 'wb') as csvfile:
    data_writer = csv.writer(csvfile, delimiter=',', quoting=csv.QUOTE_MINIMAL)
    data_writer.writerow(['Team', '1st_Downs_By_Penalty', '1st_Downs_Pass', '1st_Downs_Rush', '3rd_Down_Conversions', '4th_Down_Conversions', 'Defensive_Touchdowns', 'Field_Goals', 'Offense_Avg_Yds', 'Offense_Plays', 'Passing_Attempmts', 'Passing_Average', 'Passing_Completions', 'Passing_Interceptions', 'Passing_Touchdowns', 'Return_Touchdowns', 'Rushing_Avg_Yds', 'Rushing_Plays', 'Rushing_Touchdowns', 'Sacks', 'Time_of_Possession', 'Total_First_Downs', 'Total_Offensive_Yds', 'Total_Passing_Yds', 'Total_Rushing_Yds', 'Touchdowns', 'Turnover_Ratio'])
    for team_stats_list in all_team_stats_list:
        data_writer.writerow(team_stats_list)
    csvfile.close()

with open('nfl_team_stats2015o.csv', 'wb') as csvfile:
    data_writer = csv.writer(csvfile, delimiter=',', quoting=csv.QUOTE_MINIMAL)
    data_writer.writerow(['Team', '1st_Downs_By_Penalty', '1st_Downs_Pass', '1st_Downs_Rush', '3rd_Down_Conversions', '4th_Down_Conversions', 'Defensive_Touchdowns', 'Field_Goals', 'Offense_Avg_Yds', 'Offense_Plays', 'Passing_Attempmts', 'Passing_Average', 'Passing_Completions', 'Passing_Interceptions', 'Passing_Touchdowns', 'Return_Touchdowns', 'Rushing_Avg_Yds', 'Rushing_Plays', 'Rushing_Touchdowns', 'Sacks', 'Time_of_Possession', 'Total_First_Downs', 'Total_Offensive_Yds', 'Total_Passing_Yds', 'Total_Rushing_Yds', 'Touchdowns', 'Turnover_Ratio'])
    for team_statso_list in all_team_statso_list:
        data_writer.writerow(team_statso_list)
    csvfile.close()

```