**NFL Prediction Research**

Cody Hock – Fall 2014

For my senior project I decided to tie two totally different fields of my personal interest together: sports and computer science. In fact, I think my interest in sports directly increased my interest in statistics because I have often envisioned each game as a collection of stats that ultimately would decide the outcome. There are many so-called “experts” you see on television that will often say “forget the stats, this team is legit”, and to me this is an unwarranted argument. Yes, a team might have a lucky streak of winning games that they had no business winning based upon the statistical performance, but I believe the stats will catch up to them. I wanted to research whether a computer analyzing the statistics could be a legitimate approach to support my hypothesis.

Once I had this idea for my project, I needed a sport to test my claims. I played my fair share of sports growing up: baseball, basketball, soccer, and football. To me though, football is the one sport that I can say truly changed my life. Heck, I had offers to play in college that would have taken me away from NMU and likely towards a different career. I chose to stop playing and come to NMU for what I believe to be the right reasons (though none of which are relevant to this project). Anyways, for these reasons I decided to do some research towards determining whether only using previous statistics could create some form of predicative application for NFL game outcomes. As you will be able to see from my senior project, prior statistics have a great deal of predictive power in determining the winning team.

Overview:

The first step towards getting a prediction model is obtaining data to use in its creation. Now, there are many sites that allow users to simply download documents that contain all of the stats that the site has to offer. But this is technically a computer science course here at Northern so I thought I would be better suited to write my own custom regular expressions to pull the exact stats that I wanted from the web. Thankfully, the website at http://www.pro-football-reference.com have multiple pages of clear and concise tables of statistics for every team dating back to 2000[[1]](#footnote-1). This allowed me to write Reg-Exes that opened each page based upon the year and pulled the same groups of data year after year so that my datasets would be consistent. Though the code to accomplish this task is not long in line count, it was some of the densest code that I have written here at NMU. I chose to use PHP because it was taught in my Advanced Web Programming course last winter and we covered the topic of regular expressions. This is my longest pattern I had to match:

$passdef\_pattern = "|<td align=\"right\" csk=\"(\d\*)\">.\*\n.\*<a href=.\*\">(\D\*\d\*\D\*)</a>.\* \n.\*>(\d\*)</td>\n.\*\n.\*\n.\*>(\d\*\.\d\*)</td>\n.\*>(\d\*)</td>\n.\*>(\d\*)</td>\n.\*\n.\*>(\d\*)</td>\n .\*\n.\*\n.\*\n.\*\n.\*\n.\*>(\d\*\.\d\*)</td>\n.\*>(\d\*\.\d\*)</td>\n.\*\n.\*>(\d\*)</td>\n.\*\n.\*\n.\*\n.\*\n.\*>(-?\d\*\.\d\*)</td>|";

Any programmer with a bit of experience can see that there opportunities to shorten this pattern. Such as when I hardcode multiple lines of “junk” (\n.\*\n.\*….). However, when I use the tool to specify how many of these sub-patterns to match, the entire pattern fails to match even a single item. Oh, the joys of regular expression testing! It took me a few hours to realize what was causing my errors until I became so frustrated that I specifically wrote out all of the new line characters that I wanted to pass until the next bit of desired data. Later on I determined this was due to the fact that I did not actually want to capture the new lines, which is what I am saying with my other matches that are in parenthesis. So it was the parenthesis that doomed me from shortening my code (i.e. “(.\*\n){4}”) would not work because ‘()’ means I want to capture whatever corresponds to that segment). With more experience in reg-ex I am sure that I could clean this up a bit, but once I got my PHP working I was too afraid to make any changes that might break it.

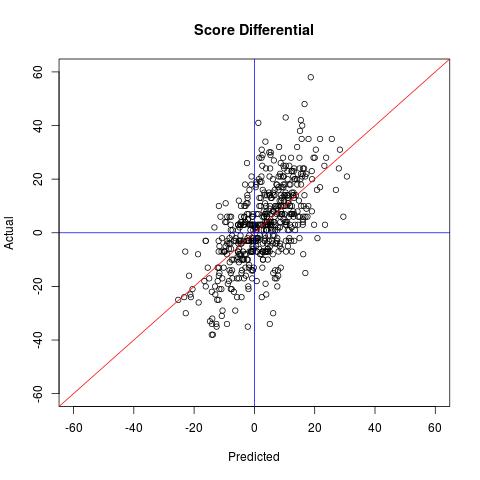
The way I wrote my PHP script allowed me to dump my pattern matches into their own .csv files. From there I could directly load them into a MySQL database consisting of multiple tables keyed on team name. This would normally be the best approach in order to keep the number of columns small for each table. However, I knew that I was going to want to load all of my data at once later on without complex SQL queries so this database schema was not going to work in this instance. Instead, I wanted all of the yearly team statistics to be placed into one table which required a little more thought up front. This is where I really got some validation from my PHP script’s output because it all lined up: there are 32 teams in the NFL currently, but this has only been true since the Houston Texans became a team in the early 2000’s. However, the number of rows in each .csv file were equal to all others from the same year. So, I was able to write a C# program that would build a new .csv for all of the team stats by looping through the smaller files and appending their contents when appropriate. Upon running that program I was left with 2 .csv files and ultimately 2 tables in my database: all of the results of games played since 2000 (scores), and the accumulated statistics for each team for each year dating back to 2000 (stats).

It was at this point that I spent a lot of my time thinking about next steps. There are a lot of great tools out there for statistical modelling and similar things. But, I wanted to do my best and pick one that would require more of a programming background than others. It was actually in an interview I had with Boeing that one of their software engineers recommended R, an open-source language specialized for statistical computing and graphics. As a bonus, I learned R supports database connections and queries. The only downside to this was I have had no prior knowledge of this language and needed to get my hands on it as soon as possible in order to learn it. Though I did not reach the desired 1,000 lines of code with my final implementation of my R scripts, I did write well over twice the amount of lines that you can see in my hand-ins in order to learn the language.

Once I had a reasonable understanding of R, I was able to start on getting my data loaded in. making the MySQL connection was straight forward and it allowed me to manipulate the data in RStudio as opposed to writing complex SQL queries. The most important aspect of my data molding was getting a correlation between my two tables. Ultimately, I decided a large data frame was necessary to hold everything I wanted. Each vector within this data frame contains a game played (with its results) from the scores table and the two teams statistics from the stats table. I did this by getting everything from my scores table and looping through each game and appending the proper team’s statistics with a 1 row resulting SQL query for that team in a certain year. Being as this was a straight forward approach, I did not anticipate any problems. However, getting the teams in the proper order was a bit confusing in R based upon how I organized my scores table. I knew I needed the home and away teams to appear in a certain order in each vector within the data frame so the machine learning algorithms later on could identify the result of the game. This was accomplished with some checks within the loop of scores, and I thought all would be fine. That is until I attempted at making one data frame for the entire data set. It turns out, with how large my table were, that this exceeded the 4GB of RAM allocated to the RStudio session. Of course this was a simple fix, but interesting never less.

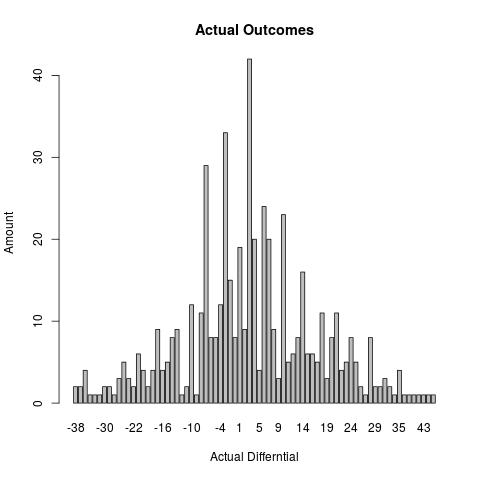
The way I organized my software was to keep a new R script that was dedicated to each algorithm I used. These algorithms were: (Multiple) Linear Regression, K-Nearest Neighbors, Decision Trees, Support Vector Machines, and even an attempted Neural Network. Consistency for comparison was achieved by using the same games (years 2000-11 as training and 2012-13 as testing sets) to model and predict. The stats used included stats grouped on the basis of: Defense, Quarterbacks, Rushing, and Special Teams. Each of these algorithms required some work to change the shape of the input data, but we will skip those descriptions for the fact that this was not a desired outcome of my project though it did take its fair share of programmer time.

My first study was to run a variety of linear regressions over the data I had collected. Regression focusses on identifying the relationship between a single numeric dependent variable and one or more independent variables. The regression function in R allows the user to enter the desired independent variables to test against the dependent and returns a model to be tested. The output of this model on the training set gives a level of significance for each statistic used in relation to the dependent result variable. Also, R allows for correlations of input fields instead of just a free standing column in the data frame on its own. This means that I was able to directly connect the stats of one team to those of the other (i.e. one team’s rushing yards gained versus the other team’s rushing yards allowed). From there, I took the input fields with the most significance (actually smallest value when it comes to levels of significance) and combined them into a larger regression. In this way I could see what combinations of team statistics could be used to gain a greater accuracy. Once a model was formed, I could use the model to predict against the testing set and compare my results with the actual results that occurred. With this larger regression model, I was able to achieve prediction the winning team 74.06% of the time. Below is a graph that illustrates how my model predicted the games in the testing set.

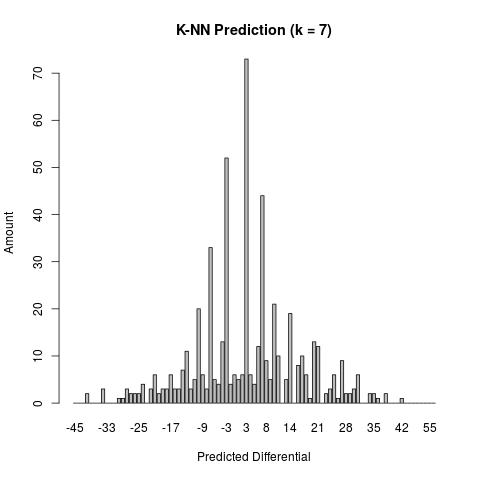


Graph 1. Shows the relationship between observed and predicted differential over the testing set.

The next algorithm I tested was K Nearest Neighbors. This differs from linear regression because there is no longer a direct formula created from a certain input to an output. Instead, this becomes a classification problem that is defined by its characteristic of classifying unlabeled examples by assigning them to the class of the most similar labeled examples. In addition to trying different stats as inputs, there were two other aspects to keep in mind for KNN that I did not have to do for regression. First, the data needed to be normalized so that one certain column does not weigh more heavily in the outcome. For example, if one column has total touchdowns in the 20-30 range and another has total yards near 3,000 the total yardage statistic will take over every classification and ignore touchdowns. The second was picking the most optimal K for polling. K represents the number of records in the training data that are the “nearest” in similarity and the test is assigned to the class of the majority of the K nearest neighbors. I tried a variety of inputs with a range of K from 3 up to 50 and found something very interesting: the most accurate model was created with just about the same input columns as my regression tests. The accuracy achieved was 71.99% for picking the correct team. However, as we can see in the graphs below, as the predictions were plotted they grouped themselves into certain results. This can most likely be attributed to the fact that KNN can result in clustering of output nodes when they should be a few units away.



Graph 2. Shows the frequency of point differentials observed in the testing set.



Graph 3. Shows the clustering observed with KNN over the same testing set.

My third choice for modeling was to use Decision Trees. This method is similar to KNN because a choice(s) must be made in order to wind up at a classification for the result by creating a tree structure with decision nodes, branches, and leaf nodes. A decision node indicates a decision to be made based upon an attribute and the proper branch path is taken. This pattern continues until a leaf node (or terminal node) is reached to denote the result of the combination of decisions made. This method was interesting to compare to KNN. I ran both algorithms with the same normalized inputs as one another, and consistently saw the same thing: KNN performed better. Though decision trees might not be the best method due to the amount of entries for each vector, it still performed rather well. After using various combinations of team statistics it was able to achieve a success rate of %70.49. The output of this was another vector filled with predicted score differentials as results that could be checked with the observed results. Unfortunately, like the previous predictions, it had a tendency to cluster at certain values and prove itself ineffective for predicting precise outcomes[[2]](#footnote-2).

My final method for prediction was Support Vector Machines, or SVM’s. In R, the call to model the SVM handles normalizing the data which was a nice feature. SVM’s handle the topic of prediction modelling a bit differently than the earlier approaches. It looks for the optimal separating hyper-plane, called the Maximum Margin Hyper-plane, between the two classes that the predictions lay within. The points that are the closest to this generated hyper-plane are called the support vectors and their distances are used to compute the plots of predictions to be determined. In order for the SVM algorithm to work, the programmer must specify what model to use such as: radial basis, polynomial, hyperbolic tangentsigmoid, or linear. The most accurate for my predictions was the “vanilladot”, or linear relationship, with an accuracy of %73.31. Using the linear option for the SVM makes this method seemingly a combination of linear regression and nearest neighbors as used above. Yet, the standard linear regression proved to be the most effective.

It can be seen that there are various ways to go about predicting the winning team in an NFL game. In addition to the methods, the combinations of statistics used are seemingly limitless. Regardless, I believe my work this semester offers some legitimacy towards this complex problem. For example, if you were to pick the home team you would have been correct about 57% of the time. Going further, picking the team with the better record would net you about a 64% success rate (teams with equal records played 11% of the time. Earlier I showed that my Linear Regressions achieved picking the winning team over 74% of the time. However, this does not mean a profit can be made from these models. The winning team is not the bet that Vegas will take. Instead they implement the idea of a point spread and that team has to win by at least that amount for you to win the bet. After the house takes its share of earnings, a better would have to be successful over 52% of the time in order to make money. Even my best model only achieved about a 39% success rate for accuracy of the correct score in the testing set. This information simply shows you will lose money in Vegas betting based off of this study (so please do not try).

Looking back, I definitely learned a lot from this project[[3]](#footnote-3). It had both the theory and application side of computer science fundamentals that can be found in most of the upper level courses in our department[[4]](#footnote-4). The hardest part of the project (or at least most frustrating) was keeping track of the current state of the data frame. RStudio allows the programmer to select which lines of code to run within a script, and this edits the variable within it. After running a certain code segment, I needed to slowly work my way thru what was going on because data what being moved around. If I ever lost track and wanted to start over I needed to run the same code in the same order all the way up to where I wanted to make a change, and this was infuriating after the tenth attempt. Obviously now I know what works and the code to accomplish that is very short, but I needed to write a great deal more in order to get to that point. So, although the length of code does not represent the amount of work I did, I am very proud of what I learned during by doing this project.

1. It must be said that what I did to get the data from https://pro-football-reference.com violates its terms of use. However, I was able to call the president of the company as well as obtain his consent in an email to use their data. This is on the condition that I do not propagate my database for anyone else and all of my research be purely educational (i.e. do not make a working predictor that can be sold and monetized off of pro-football-reference.com’s content). [↑](#footnote-ref-1)
2. It is interesting to note where the clustering occurring in KNN. The most common predicted outcomes were -7, -3, 3, and 7. These are of course the findings we would see if one of the teams were to win by either a field goal or a touchdown with an extra point (which are the most common point totals awarded for scoring plays). [↑](#footnote-ref-2)
3. One thing I cannot forget is the importance of source control. At one point during this project I accidentally deleted my one and only copy of my PHP script after it was already completed. Luckily, it was stored on Euclid and Randy was able to perform back-ups wizardry. At that point I set up a github.com account and all of my code is safely stored in my own repository there. [↑](#footnote-ref-3)
4. I mentioned I attempted a Neural Network earlier. I was able to get the data input to the function properly, but I never had sufficient time to analyze what the output was. This was my last method attempted, and since that time I have been overwhelmed with the stresses of post-graduation job interviews and deemed them more important. [↑](#footnote-ref-4)