Capstone Report for NFL Data

Abstract:

There are two objectives to this capstone. The first one is to see if there is any factor significantly correlated to injuries lower limb in the NFL, such as a specific type of weather or stadium type. The second objective is to correctly predict on the test set whether a lower limb injury will occur or not. **This sentence should then discuss the results of our models.**

Table of Contents

Mission Statement:

There are two objectives to this capstone. The first one is to see if there is any factor significantly correlated to injuries in the NFL, such as a specific type of weather or stadium type. The second objective is to correctly predict on the test set whether an injury will occur or not. The information was obtained from the competitive coding website, Kaggle, and was provided by the NFL for a competition to see who could build the most accurate predictive model.

The data consisted of three different tables: playlist (267,005 data points) which focuses on stadium type/ weather conditions/play type/ field type , trk (76,366,748 data points) which focuses on the motion of the individual players for every tenth of a second giving us position and velocity, and inj (105 data points) which focuses the injuries that occur, specifically, providing information play key, player key, surface, body part injured, and the duration the injury kept the player out of the game.

Using these data sets, I plan to clean, feature engineer, and resample the data to draw conclusions on correlation and create a predictive model to see if we can predict the next injury! Personal note, this is kind of scary and ominous if you think about it.

Cleaning Data:

There are two dark horses I want to address directly first. The weather and stadium columns of the playlist data table were awful. Talking solely about stadium, I had to look up the unique values of that data set to label and create a categorical column of either indoor or outdoor, based on the unique values and my knowledge of Heinz field (a bowl type stadium for the Pittsburgh Steelers) using the .str.contains function. Then I had to do the same for the extremely varied weather column. This is where some feature engineering came into play. Based on the unique values I extracted on the data set I made 4 new columns out of the weather being snowy, rainy, sunny, and cloudy. I would have liked to create a windy column, but I did not think there was enough information to support it.

The next dark horse created a lot of problems when merging and that is that the play value was missing for some injuries. As if there weren't few enough injuries already in the injuries data set (105), now there were even fewer (77) that I could actually analyze. A potential issue with this in the future is that the play was recorded in the playlist data set, but it is impossible to determine which plays in the injuries data set that were unlabeled actually resulted in injuries. This throws a huge wrench in plans for undersampling data, since if we unfortunately end up with one of these 27 mislabeled plays we could unknowingly create a bad model. This means we that the safest way to go about resampling is to oversample, but I am getting ahead of myself. Going back to the point before, there are only 77 injuries we could realistically analyze when joining the playlist and injury data sets. Which allows me to draw conclusions about correlations and other data analysis based concepts.

Now that some feature engineering has happened and we have cleaned the data a bit, it was time to make a master data set, I thought. As nice as it would be to combine all three datasets into one master dataset, it just is not practical computationally. It makes the most sense to merge all three data sets into two different ones, because of the trk dataset primarily. It has a lot of data points since it has a data point for each play for each player for each tenth of a second, which would mean that not only would be it unnecessarily computationally expensive to run even basic analysis, but there would be a lot of data points that encompass a play that would be repeated. These repeated data points would have to be filtered out and, although possible, is also computationally taxing and is just room for error. The best solution here is to just create a table merging playlist and inj and then another merging trk and inj.

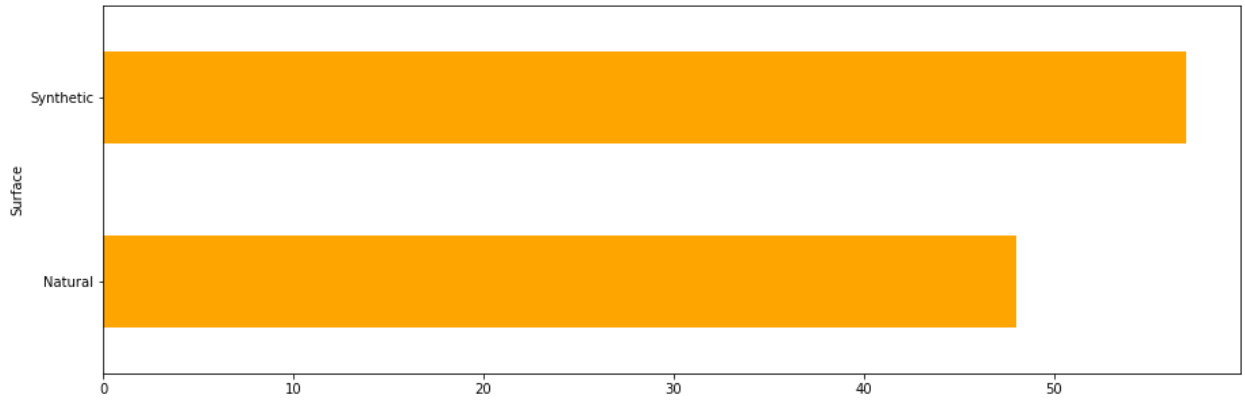
How I merge here matters. Depending on the information I want to extract I will be merging the tables differently. For example, if I just want to see more information about whether a play led to an injury or not, then I would have the inj table merge onto playlist table. However, if I just want more information on the injuries, then I would merge the other way around. So it is best to make two tables out of our two tables. Funny!

First Impressions:

The table trk is going to be the biggest problem. How I manage the information in this table is important for progress since RAM is a plenty, but not infinite. If I want to run some basic analysis functions I am going to have to play around with some of the columns and change them to either type int or boolean. For whatever reason there are some values that have been mislabeled in the data such as temperature being 999 degrees. For a data set as large as it is, I can just omit these values altogether. Since most of the values in my data are going to end up being boolean, I am going to have to normalize over any column of type int so as not to get any weird weighting. As mentioned before, there is also extreme bias going on in this data set and probably a lot of variance happening in our smaller class, so resampling methodology is going to have to be utilized to account for this issue.

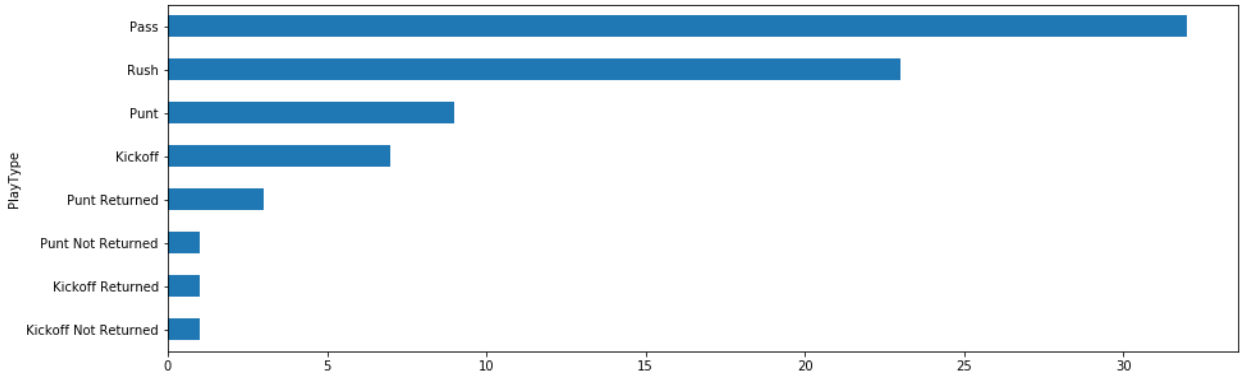
Statistical Analysis:

There is a lot of unpack here. So let me begin with analyzing the inj table.



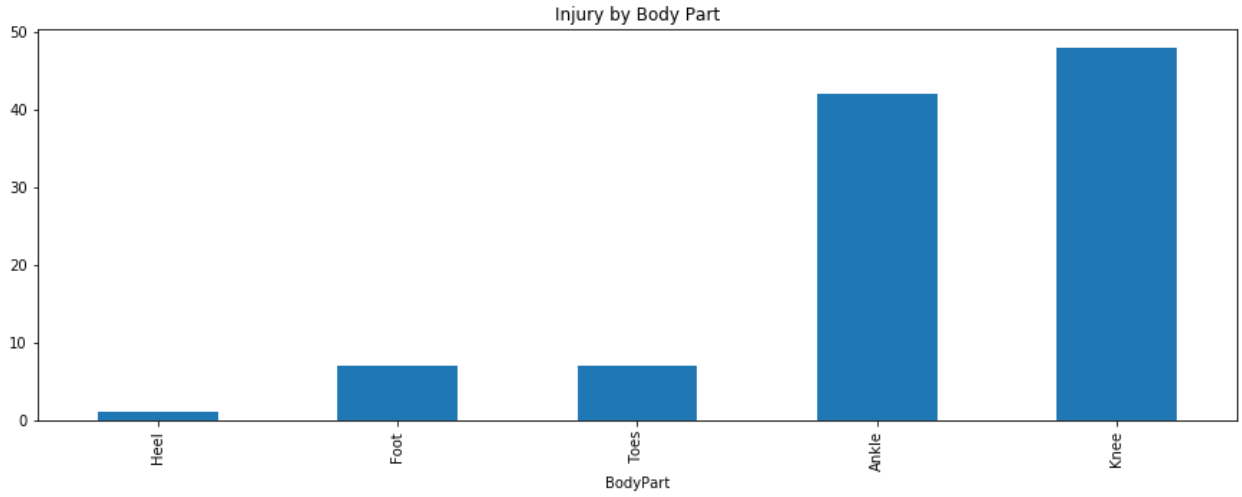
This table shows the type of grass associated with a play that resulted in an injury. For the amount of data points that we have, we can see there is not a huge statistical difference between the two. In fact, there is only a 7% difference between the two.

Next, I want to look at if certain plays are more likely to lead to injuries. So we can make a chart that shows all the of play types that led to injuries.

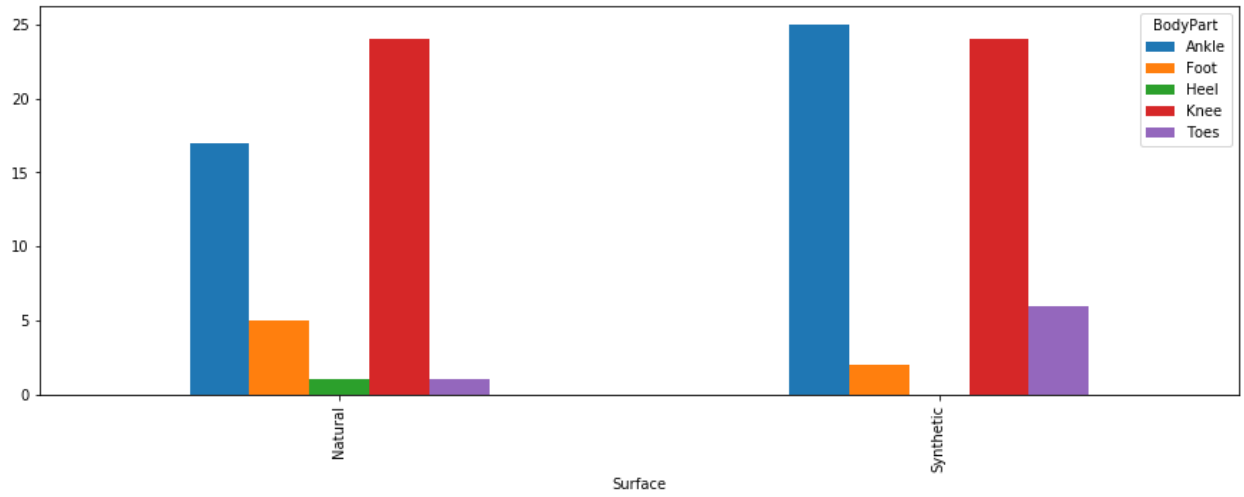


Although this looks promising, we need to look at the distribution of play as well. If there truly is a relationship between the type of play type and injury likelihood, then there should be a disproportionate amount of injuries on that play compared to the total fraction of plays of that type. The fractional difference for the top two plays is 19% and 13% respectively. Doing further analysis on this is difficult due to the incredibly small amount of injuries for the other plays. It would just be wild and reckless to conclude anything off of >10 data points in a data set of our size.

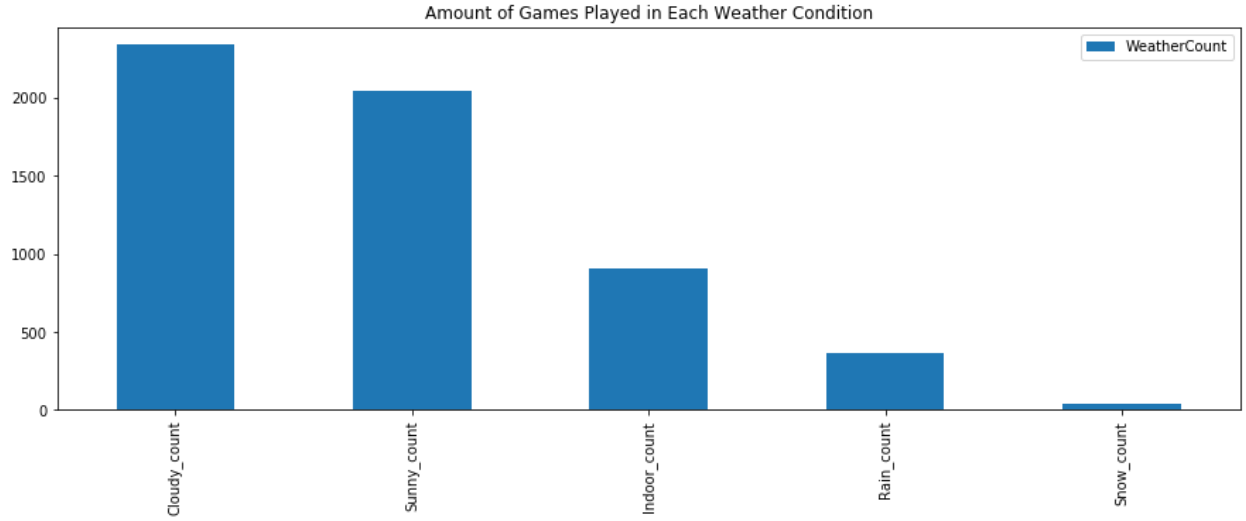
The next chart is just to get a better understanding of our minority class and what information it contains. I do not want to evaluate and make conclusions based on too few data points. Also it could lead to utilizing proper under and oversampling technique implementation.



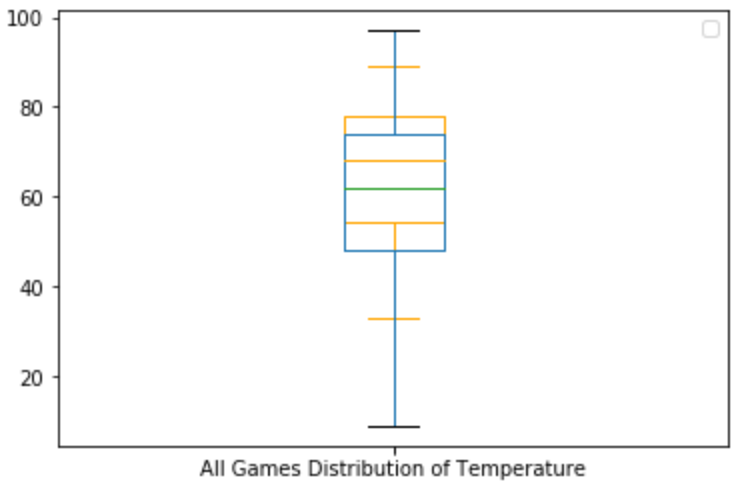
A quick graph showing the injury type dependent on the type of grass could show us a lot in a small graph. Not only could it be revealing if an injury is more likely dependent on the type of graph, but also if there is a disproportionate amount of injuries in one category than the other, then we could say that there is an existing relationship between that injury type and the grass.



Unfortunately for me, fortunate for the football players, there is no huge conclusion to be made here. Although 25% more injuries happen on the ankle when playing on synthetic vs natural, it is only 7 more instances in a data set of 105. Not a huge difference. Difficult to conclude anything.

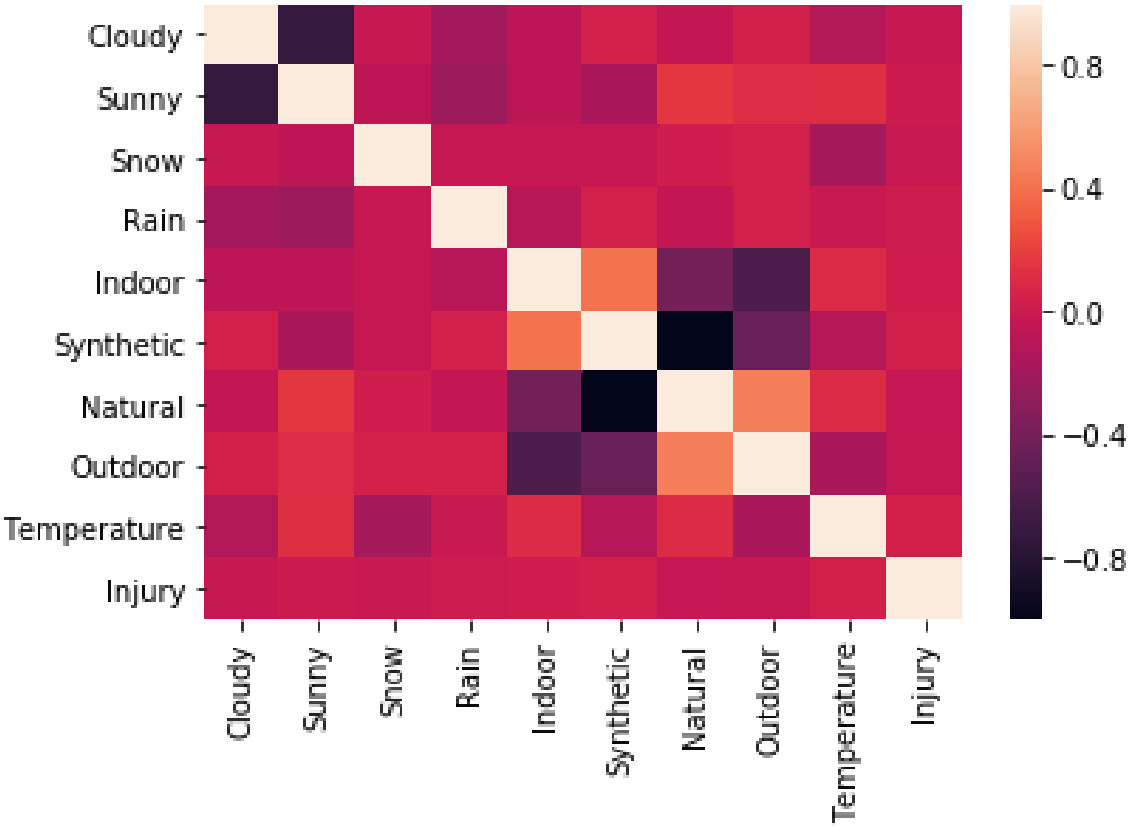


Maybe weather could be a factor. Had to extract the information from the weather column. The result was the table above.



Want to see if maybe there is also some huge shift in temperature leading to injuries. The difference between the means of the two is only a few degrees. As expected.

Finally, lets just see if these factors have any correlation by running a correlation function.



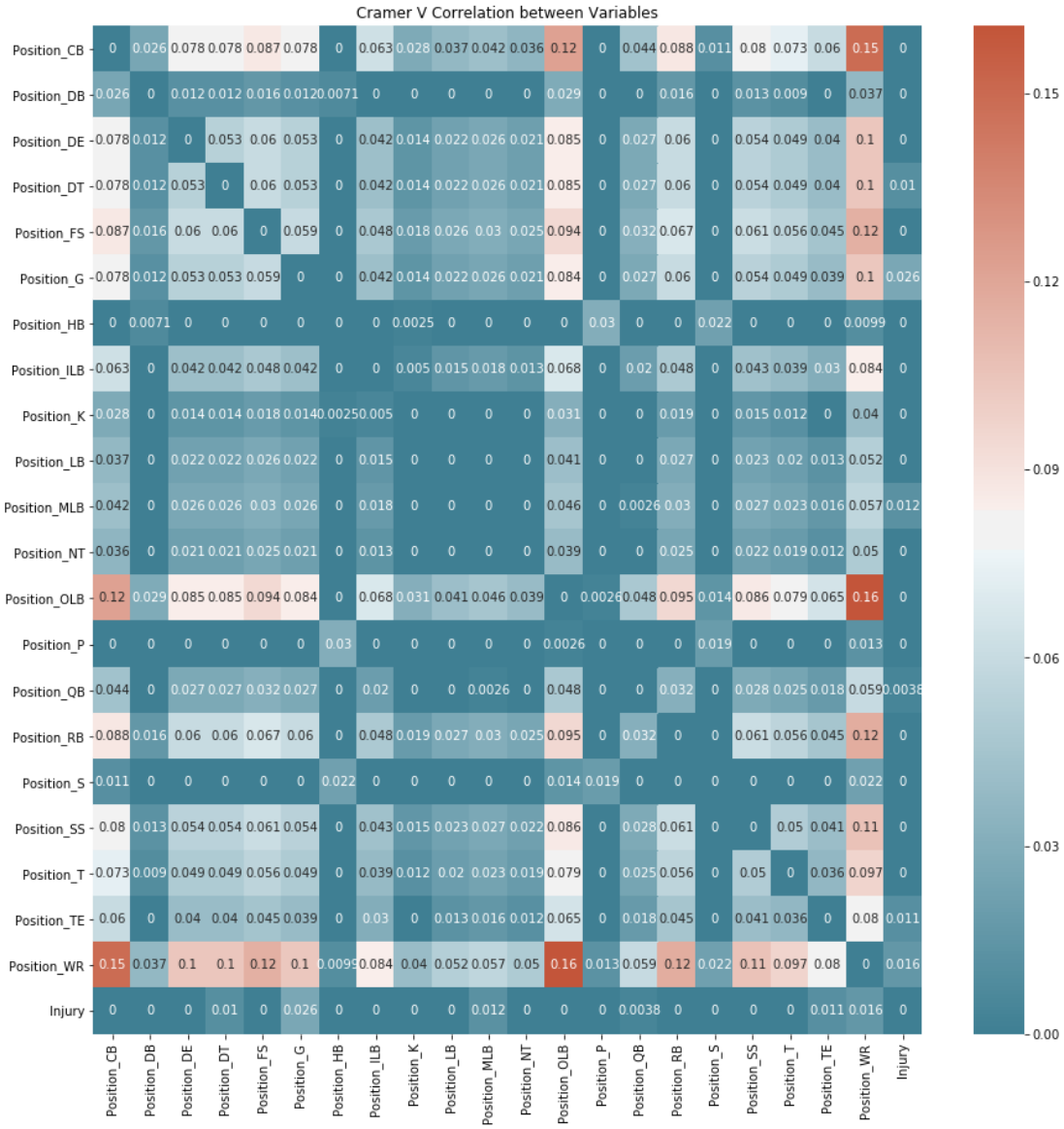
There are some obvious results here that enforce what we know and show us everything is working correctly. Synthetic and natural should be extremely negatively correlated since a field type cannot be both. Synthetic and indoor have positive relationship which also makes sense. Although there are no immediately correlated values to injury, this is not all bad. Some machine learning models take these weakly correlated factors and use them to make a strong algorithm. If that becomes the case, I may need to exclude some of the correlated factors so as not to double dip into the same relationships.

I found this method that determines how important a relationship is for determining the outcome. For example, if you are trying to determine if a field is synthetic or not, if you know that the field has natural grass, then you know for sure what kind of grass it is. This is why the Cramer V value is 1. Similarly, the relationship between itself is for all categories, because it is assumed that if you are trying to determine an outcome, knowing the factor of itself is not important, so it should be zero since there is no information gain.

Nothing but extremely weak factors, some even resulted in zero significance in the weather category. Strange to think that rain offers no information as to whether or not a player will be

injured. This does help in excluding extraneous factors when applying to our machine learning model.

Running the function on position to see if there is any important correlations with position and injury yield some interesting results. From these potential results we can infer some other bits of information by knowing what the position entails. For instance, if linemen are more likely to be injured than other positions, then we can infer that extended contact could be a huge contributor to injury since it is the most physical position. Or if the position most important turns out to be wide receiver, then a potential factor to look at would be the max velocity of the player on the play since they tend to be the quickest player.



Preliminary Model Training:

(Something to keep in mind throughout this portion is that all the tests that were ran were ran on resampled test sets and NOT on the original sets. This makes the results of these tests are only indicators of potentially good methods, but by no means are accurate indicators of how well the model actually performs. I chose not to correct this, because it is only preliminary testing and not my final tests.)

There is the obvious issue of class imbalance in the data set. With only 105 data points for injury and 26000+ data points that do not have injuries associated with them, there is an immense amount of bias that needs to be accounted for to train a decent model.

Firstly, I want to define how well my model does. Using strictly accuracy, we'll produce a "good" model since it's score would be in the 90% range, but its utility would be complete garbage. To exemplify this, I ran a dummy model that would predict on every point that the point is part of the majority class and got a 97.5%. I ran a ADABoost model on the data raw as well and got the exact same score down to the millionths place. They literally predicted the same thing, so not only do I need to establish the right metric of accuracy, but also apply some bias reduction.

Since what I care about is the total amount of true positives, recall is the best unit of accuracy. I only care if I can correctly predict when someone is going to be injured. Precision does as well serve some form of utility, so maybe an F1 score would be useful, but, for now, recall does the job for initial training.

Next is applying some bias manipulation. There are two different routes I can take here: undersampling and oversampling. I first applied undersampling to my models. The issue with undersampling is that even though we reduce the amount of bias on the majority class, we introduce a lot of variance. The whole bias-variance trade-off thing occurs in spades. We only pull about 0.3% of our data points in the majority class which leads to an immense amount of information loss and our model shows it. I used a random sampling of the majority class utilizing the imblearn library. I got a precision score of 0.72 and an F1 score of 0.66.

Now I want to apply oversampling of the minority class.There are three different methods to apply: random, SMOTE, and ADASYN. I will explain each along with their results. Since this is only preliminary testing, I will only run each resampling method against one good out of the box machine supervised machine learning method. I am just trying to get a grip and a general understanding of their results. Once I am ready to incorporate movement, then I will run the full model (if I figure out how). Random is the simplest one of the oversampling methods. All it does is duplicate the minority class' data points. The issue with this is that the impact of any outliers has now been amplified by a significant amount as well as the overall variance of our data, but that is as expected. The results from this test were a precision of .60, recall was .50, and F1 was 0.54 when applied to the test set.

SMOTE is another oversampling method to apply to the minority class. What it does is it creates a vector in multidimensional space connecting k nearest neighbor points of the minority class, specified by the user, and randomly distributed points throughout the vector (actually defined by a lambda value between 0 and 1 that is optional) until the minority class has as many points as the majority class. One of the issues with this method is that it creates "bridges." When

there are outliers standing outside the obvious clusters, the point will produce a line of fabricated data points that connect it to the main cluster. It looks weird and can be overall damaging to our data, but if it helps the overall result, then worth. I first resampled, then applied train test split. Significantly better results arose from this with precision of .71, recall of .68, and F1 of 0.70 when applied to the test set.

ADASYN is a bit more mathy than SMOTE. With ADASYN we start by determining to what degree of imbalance we will allow in our data by taking the ratio of the two classes.

$$d = m_s/m_l$$

Where m_s is the minority class, m_l is the majority class and d is the resulting class imbalance. If this value is less than what we want, then we run a series of functions to determine where to generate data points until we have reached the desired ratio.

Next, to determine the number of data points to generate we pick a value for beta between 0 and 1 and calculate:

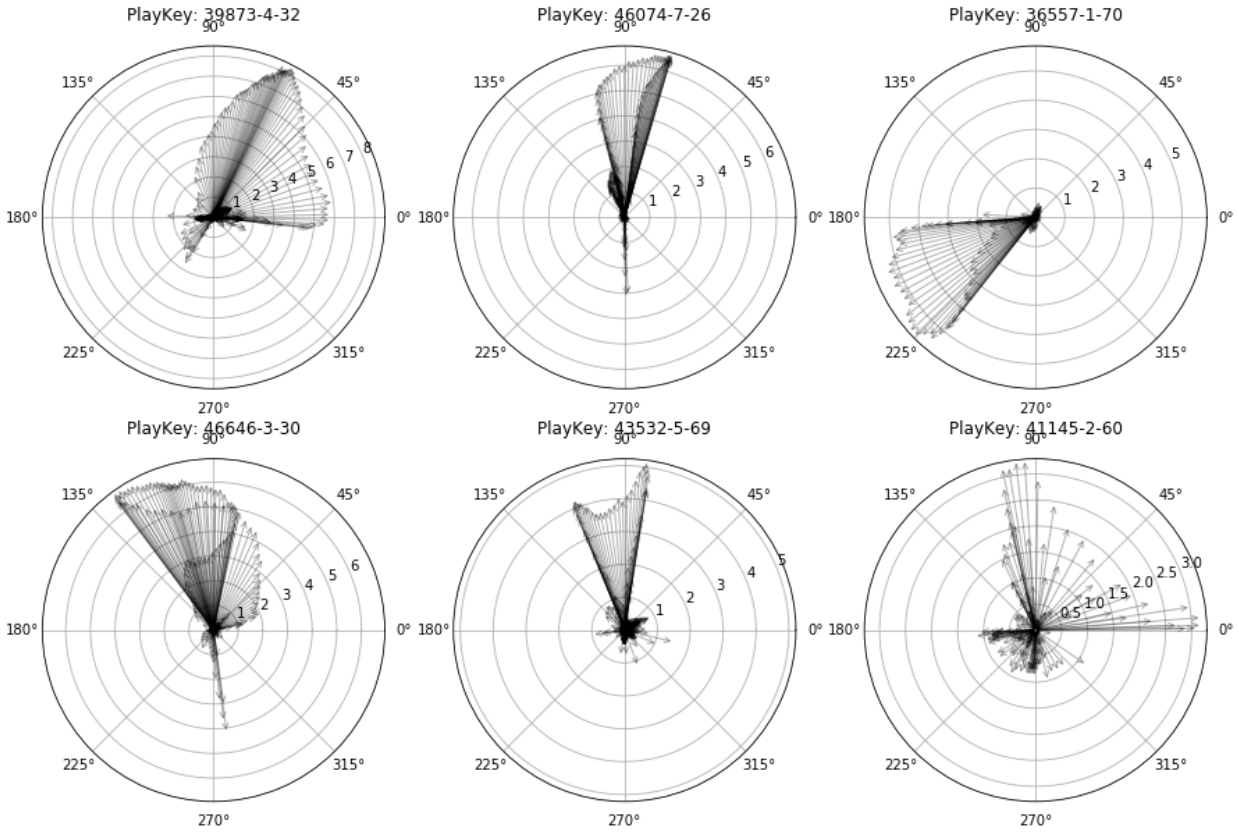
$$G = (m_l - m_s) \cdot \beta$$

Then for each data point of the minority class we determine K nearest neighbors based on euclidean distance and calculate a ratio of majority class points over K for each. Then normalize by taking the sum of all these ratios and dividing each point by the sum. We then calculate the number of points we need to generate in each neighborhood by multiplying each of these normalized ratios by the G we generated earlier which makes sense since the normalized ratio's sum is 1. Doing this over all points will result in exactly G number of data points generated. We then randomly place number of data points we need to within each neighborhood.

Only thing left is to observe the results of this resampling method. Across all score types previously mentioned it scored 0.72 on the test set. Impressive! Damn shame that I resampled the test set, otherwise this could have been insanely great results.

More Analysis:

This section briefly explores the motion data set and the actions that occurred on the plays where someone was injured. Utilizing the code from another Kaggle competitor, the following beautiful charts were created tracking direction of motion and the speed of the player being represented by the magnitude of the arrow.



I just thought that it was a really cool visualization technique and want to be able to do this myself one day. It is a bit misleading though since all the compass plots have different markers determining the speed. Difficult to visually compare, but easy to understand. Perhaps another improvement that could have been made is to add a color gradient deterministic of the amount of time elapsed during the play to kind of show the direction of the path taken. Given more time, I would have attempted to implement this myself.

Speaking of time, there is tons of potential feature extraction to be exercised here, but, unfortunately, time is a factor. Given more time, the features I would have loved to extract from each play is max velocity, average velocity, greatest change in direction given some small time frame, and maybe the greatest change in velocity in some small time frame.

Final Model Training:

Time for the big boy modeling and proper resampling and data splitting methods. So there is a very specific order in which I need to execute the resampling to get proper results that accurately reflect how well my model can predict injury. First, I separate the predictive factors from the result. Then I initialize the stratified K-fold function so as to split the data set into a test and train set while maintaining the ratio of positive readings to total readings among both sets. Then I normalize all my factors and make sure I use the .normalize function in order to account for the different units that are present in all the different columns. It is only really necessary

because of the temperature column, but still necessary. Now that I have all of my data set up, I can go through each of the resampling methods and apply different machine learning models.

Utilizing the same out of the box Adaptive boost classifier function, I tested it across all the previously exercised resampling methods. ADASYN performed the best, so we continued on with that resampled data set.

Next, I applied the resampled data set to 5 different supervised machine learning models and decided I would just parameter tune whichever performed best. It turned out AdaBoost did the best, but logistic regression was a close enough second that I thought it worthwhile to investigate as well. From there I used gridsearch to tune a couple of the parameters that are known to have the highest impact for their respective model. I got something interesting from the AdaBoost. I was able to predict all injuries perfectly with a recall of 1 at the expense of everything else. So we were misclassifying a huge portion of the data set, but this is where the goal comes to mind. At the end of the day, it becomes a question of what is more important. Is it more important to predict injury when it happens? Is it more important to not misclassify? Or is there a harmonic mean we would be happy with?

The following results for the max value of each given our algorithms ordered precision, recall, F1 and against the training set first, then the test:

Max Precision: AdaBoost n_est = 1 l_rate = 1
[0.98628049 0.03322785]
[0.76057994 0.4375]
[0.85884956 0.06176471]

Max Recall: AdaBoost n_est = 1 l_rate = 10
[0. 0.01846154]
[0. 1.]
[0. 0.03625378]

Best F1-Score: AdaBoost n_est = 1 l_rate = 1
[0.98628049 0.03322785]
[0.76057994 0.4375]
[0.85884956 0.06176471]

Conclusion:

Throughout the course of the project it is important to keep in mind the goal. We want to predict whether a player or not will be injured on any given play depending on factors such as environment and motion. With that in mind, I was able to take the immense and unorganized data set and turn it into a workable set of tables with combined factors to better understand what was happening. As well, I dealt with outliers through mostly the removal of such data points which were clearly the result of documentation errors (temp at 999 degrees fahrenheit). I then

resampled the data in such a way as to get the best results possible from my machine learning algorithms and ended with the following results.

The following results for the max value of each given our algorithms ordered precision, recall, F1 and against the training set first, then the test:

Max Precision: AdaBoost n_est = 1 l_rate = 1
[0.98628049 0.03322785]
[0.76057994 0.4375]
[0.85884956 0.06176471]

Max Recall: AdaBoost n_est = 1 l_rate = 10
[0. 0.01846154]
[0. 1.]
[0. 0.03625378]

Best F1-Score: AdaBoost n_est = 1 l_rate = 1
[0.98628049 0.03322785]
[0.76057994 0.4375]
[0.85884956 0.06176471]