

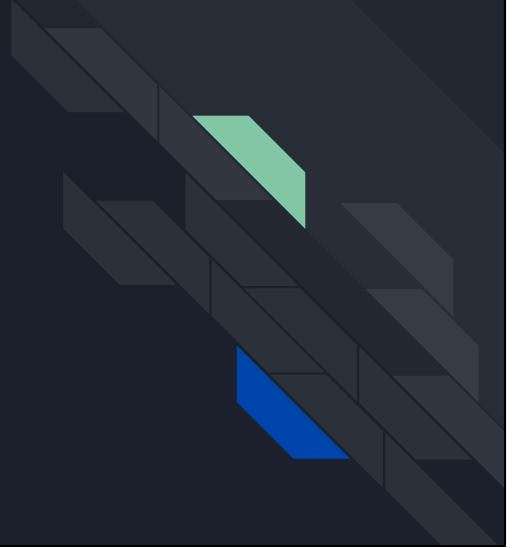


NFL Injury Prediction

Kushal Gourikrishna
DATASCI 207 Final Project

Objective

Build a classification model that can help predict injuries to NFL players based on a suite of factors present during a typical NFL game





Injury Data

Injury Data File

Field	Format	Description
PlayerKey	XXXX	Uniquely identifies a player with a five-digit numerical key
GameID	PlayerKey-X	Uniquely identifies a player's games (not strictly in temporal order)
PlayKey	PlayerKey-GameID-X	Uniquely identifies a player's plays within a game (in sequential order)
BodyPart	character string	Identifies the injured body part (Knee, Ankle, Foot, etc.)
Surface	character string	Identifies the playing surface at time of injury (Natural or Synthetic)
DM_M1	1 or 0	One-Hot Encoding indicating 1 or more days missed due to injury
DM_M7	1 or 0	One-Hot Encoding indicating 7 or more days missed due to injury
DM_M28	1 or 0	One-Hot Encoding indicating 28 or more days missed due to injury
DM_M42	1 or 0	One-Hot Encoding indicating 42 or more days missed due to injury

Dataset is specifically focused on lower-leg injuries. Premise of Kaggle competition was to see if turf type could impact these injuries. I wanted to expand the premise into a broader machine learning prediction problem

Play Level Data

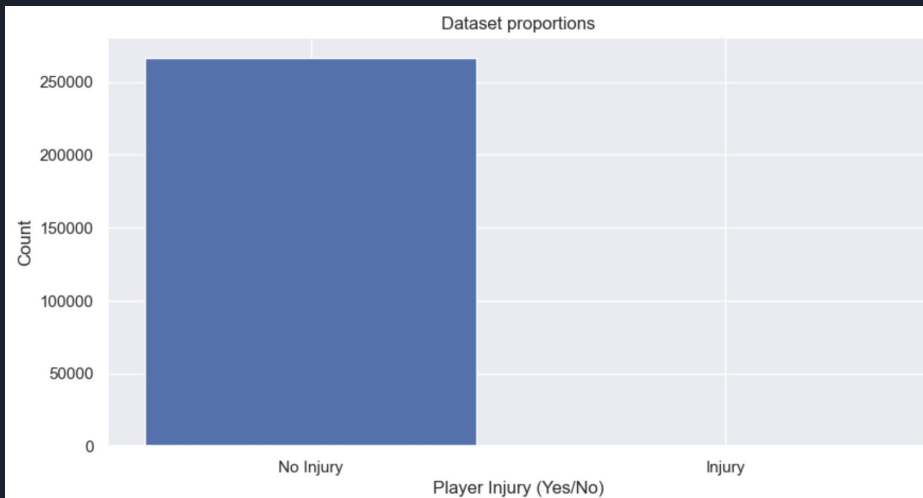
Field	Format	Description
PlayerKey	XXXX	uniquely identifies a player with a five-digit numerical key
*GameID	PlayerKey-X	uniquely identifies a player-game (this index is not strictly in temporal order) – see note below
PlayKey	PlayerKey-GameID-X	uniquely identifies a player's plays within a game (in sequential order within a game)
RosterPosition	character string	provides the player's roster position (i.e. Running Back)
*PlayerDay	integer	an integer sequence that reflects the timeline of a player's participation in games; use this field to sequence player participation
*PlayerGame	integer	uniquely identifies a player's games; matches the last integer of the GameID (not strictly in temporal order of game occurrence)
StadiumType	character string	a free text description of the type of stadium (open, closed dome, etc.)
FieldType	character string	a categorical description of the field type (Natural or Synthetic)
Temperature	float	on-field temperature at the start of the game (not always available - for closed dome/indoor stadiums this field may not be relevant as the temperature and weather are controlled)
Weather	character string	a free text description of the weather at the stadium (for closed dome/indoor stadiums this field may not be relevant as the temperature and weather are controlled)
PlayType	character string	categorical description of play type (pass, run, kickoff, etc.)
PlayerGamePlay		an ordered index (integer) denoting the running count of plays the player has participated in during the game
Position	character string	a categorical variable denoting the player's position for the play (RB, QB, DE, etc.) – may not be the same as the roster position.
PositionGroup	character string	a categorical variable denoting the player's position group for the position held during the play



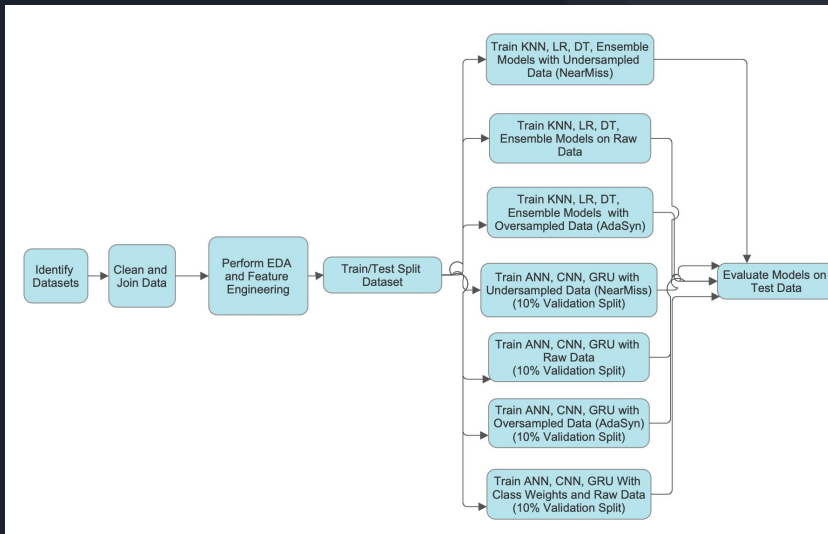
Player Tracking Data

Field	Format	Description	Additional Details
PlayKey	character string		
time	float	time in seconds (float) since the start of the NGS track for the play	this is the time index for the player track
event	character string	play details (character string) as a function of time during the play (huddle break, snap, etc.)	
x	numeric list	player position (float) along the long axis of the field (yards) over time index	0 - 120 yards
y	numeric list	player position (float) along the short axis of the field (yards) over the time index	0 - 53.3 yards
dis	numeric list	distance traveled (float) from prior time point over the time index	Distance (yards)
s	numeric list	estimated speed (float) at that particular point in time over the time index	yards per second
o	numeric	Orientation (float) - angle that the player is facing (deg)	0 - 360 degrees
dir	numeric	Direction (float) - angle of player motion (deg)	0 - 360 degrees

Imbalanced Data



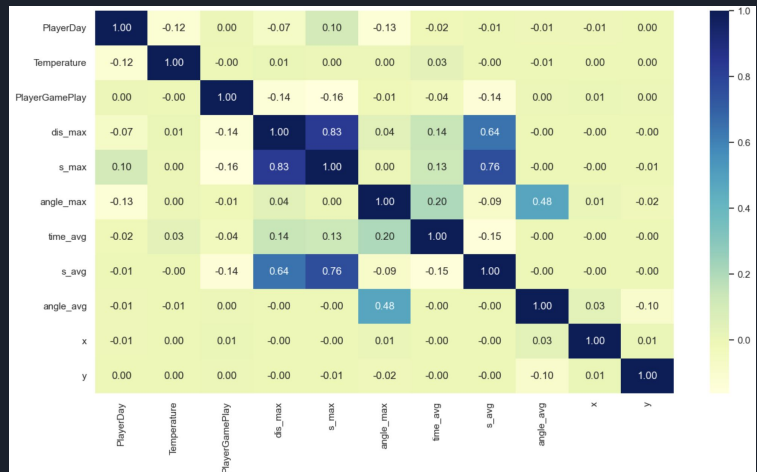
Approach and Methodology



The Train/Test split 80/20. For neural networks, an additional 10% was used as validation set (taken from training data). As seen on this block diagram, I experimented with the different data augmentation techniques across all the different model types and ultimately evaluated the raw test data on each.

Feature Engineering

- Fill missing values
- Compute average and max speed, distance, etc
- Drop highly correlated features
- Merge three datasets together to create input for models





Success/Failure Criteria

- General Rule: >70% Model Recall
- Secondary Metrics
 - Accuracy
 - F1 Score
 - Precision
 - ROC_AUC Score

Looking at success/failure criteria now compared to the midterm, my focus shifted to recall score as opposed to purely accuracy due to the imbalance of the data.

Accuracy as we know didn't really mean anything for my problem as I could achieve 99% accuracy just by predicting all majority examples. The focus really became trying to classify the minority class correctly. Some other metrics that were looked at secondarily were accuracy, F1 score, precision, and ROC_AUC Score (area under the curve score).



Model Results (Non-Neural Networks)

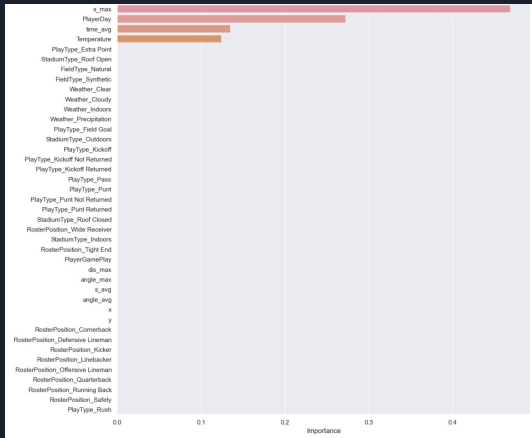
Now that we've gone through the approach and methodology, let's dive into the model results. I'd like to start with non-neural networks. These, as we will see, include KNN, Logistic Regression, and different ensemble methods.

Models Using Raw Data (With Class Weights)

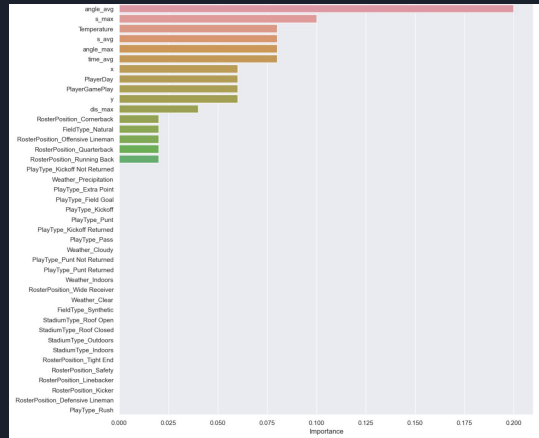
	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	KNN	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
1	Logistic Regression	0.704708	0.619048	0.380952	0.000508	0.001015	0.542894
2	Decision Tree	0.999306	1.000000	0.000000	0.000000	0.000000	0.499850
3	Decision Tree Tuned	0.463974	0.285714	0.714286	0.000524	0.001048	0.589081
4	Random Forest	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
5	Random Forest Tuned	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
6	XGBoost	0.999569	1.000000	0.000000	0.000000	0.000000	0.499981
7	AdaBoost	0.806513	0.857143	0.142857	0.000291	0.000581	0.474816

Let's first just take a look at the running these models with our raw data and using class weights. Essentially, for each of these a parameter was included to account for the imbalanced nature of the data as 'class weight'. With these initial results, we see that most of the models still overfit or simply predicted the minority class. However, interestingly we do see some different results with Logistic Regression, the tuned Decision Tree, and Adaboost. I think most notably, the tuned decision trees actually reached the desired threshold of 70% recall. However, a theme which we'll continue to see is the very very low precision and as a result F1 score for all of these results. Before we move on, I wanted to highlight two models and take a look at the feature importance.

Feature Importance



Tuned Decision Tree (Raw Data)



AdaBoost (Raw Data)

With more time I would've liked to dig deeper into the individual features and filter out important information. However, for the class I wanted to highlight these two models that produced a relatively good Recall score and see how they ranked the features in importance. We see some crossover between the two models such as "Max Speed", "Temp", "Average Play Time", and "Player Day". Adaboost seemed to consider additional features while the tuned decision tree simply used these four features.

Models Using Undersampled Data

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	KNN	0.858260	0.904762	0.095238	0.000265	0.000529	0.476899
1	Logistic Regression	0.417740	0.285714	0.714286	0.000483	0.000965	0.565954
2	Decision Tree	0.087688	0.285714	0.714286	0.000308	0.000616	0.400863
3	Decision Tree Tuned	0.060127	0.047619	0.952381	0.000399	0.000797	0.506078
4	Random Forest	0.074264	0.047619	0.952381	0.000405	0.000809	0.513149
5	Random Forest Tuned	0.048522	0.095238	0.904762	0.000374	0.000748	0.476473
6	XGBoost	0.075557	0.095238	0.904762	0.000385	0.000770	0.489996
7	AdaBoost	0.091850	0.047619	0.952381	0.000413	0.000825	0.521946

The next set of results we will look at are the models that were fit on undersampled training data. This used the 'Near Miss' method. This method selects examples from the majority class based on their (Euclidean) distance from examples in the minority class. With these results, what I found most interesting was that with the undersampling data most of the results seem to lean heavily towards the minority class. Perhaps too much as we see recall scores up in the 90th percentile but accuracy that is much worse than what we saw in the previous slide. Additionally, precision remains low. As a result, I didn't find any of the models here to be very useful.



Models Using Oversampled Data

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	KNN	0.999531	1.0	0.0	0.000000	0.000000	0.499962
1	Logistic Regression	0.006018	0.0	1.0	0.000396	0.000792	0.502813
2	Decision Tree	0.998913	1.0	0.0	0.000000	0.000000	0.499653
3	Random Forest	0.999606	1.0	0.0	0.000000	0.000000	0.500000
4	XGBoost	0.999588	1.0	0.0	0.000000	0.000000	0.499991
5	AdaBoost	0.994094	1.0	0.0	0.000000	0.000000	0.497243

Finally, in terms of non-neural network models we'll take a look at fitting the models with oversampled data. Here, we see all of the models basically overfit and predicted the majority class. Except for logistic regression which did the opposite and leaned heavily toward the minority class. A theme we'll notice is that oversampling really did not help with any of the models I looked at. With the very heavy imbalance, my theory is introducing that much synthetic data to equalize the two classes essentially provided no impact and led to overfitting.



Model Results (Neural Networks)

Now let's dive in the neural network experimntaiton



ANN Results

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	Basic Neural Network	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
1	Weighted Neural Network	0.407203	0.190476	0.809524	0.000537	0.001074	0.608284
2	Undersampled Neural Network	0.554306	0.380952	0.619048	0.000547	0.001093	0.586664
3	Oversampled Neural Network	0.998800	1.000000	0.000000	0.000000	0.000000	0.499597

I first want to look at a simple artificial neural network that I built. As discussed all of the neural networks we'll see were tested with unweighted, weighted, oversampled, and undersampled data. With the ANN here I specifically wanted to highlight the weighted neural network. Here we see a very good recall score of around 81%. Furthermore, unlike the undersampled models we saw before we are not sacrificing as much accuracy here. Nevertheless, precision remains very low but we do see a slight uptick in the ROC/AUC score.



CNN Results

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	Convolutional Neural Network	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
1	Weighted Convolutional Neural Network	0.869190	0.809524	0.190476	0.000574	0.001145	0.529967
2	Undersampled Convolutional Neural Network	0.330971	0.238095	0.761905	0.000448	0.000896	0.546353
3	Oversampled Convolutional Neural Network	0.999119	1.000000	0.000000	0.000000	0.000000	0.499756


With the last two neural network types, I would say these were more explorational. With CNNs, we learned in class that these are used mostly for images or higher dimensional data. However, you can build a CNN for 1D data so it was iterating to see the results. Here, I just want to highlight the undersampled network. Similar to the ANN before we see a recall above our threshold but the accuracy is worse and precision is actually even worse. To me this was mostly expected as the application of the CNN didn't really fit my classification problem.



GRU Results

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
0	GRU RNN	0.999606	1.000000	0.000000	0.000000	0.000000	0.500000
1	Undersampled GRU RNN	0.817256	0.619048	0.380952	0.000821	0.001639	0.599190
2	Oversampled GRU RNN	0.999156	1.000000	0.000000	0.000000	0.000000	0.499775
3	Weighted GRU RNN	0.628738	0.761905	0.238095	0.000253	0.000505	0.433494

Finally, as a last ditch effort I tested an RNN. Specifically I build a very simple Gated Recurrent Unit network which is a variation of an RNN similar to LSTM. Again as we learned in class RNNs are mostly effective for natural language processing or speech recognition. And we see that play out here with none of the model performing very well on this dataset. We don't see any reach our desired recall threshold.



Discussion

- Weighted Neural Network gave overall best performance
 - Recall (0.80), F1(0.0011), Accuracy (0.41)
 - Was not trained with augmented data
- Heavy Imbalance in data still a big issue
 - Undersampling helped results slightly
 - Oversampling mainly just led to overfitting
- Overall Precision and F1 scores very low

	Model	Accuracy	FalseNegRate	Recall	Precision	F1 Score	ROC_AUC
1	Weighted Neural Network	0.407203	0.190476	0.809524	0.000537	0.001074	0.608284

Ultimately, I felt the weighted neural network was the best performing model. However, I hope it was obvious that the heavy imbalance in the data was still a huge issue. Precision and F1 score while not the focus were still way too low to have the model be considered useful and undersampling/oversampling didn't improve our results in a meaningful way.



Limitations and Future Work

- Time and resource constraints



- Investigate individual neural network architecture and other techniques further

- Limited to Kaggle data and provided feature list



- Collect more player data
 - Increase feature list
 - Expand scope of injuries
 - More seasons, more players

Thank You!