Greg Vargas

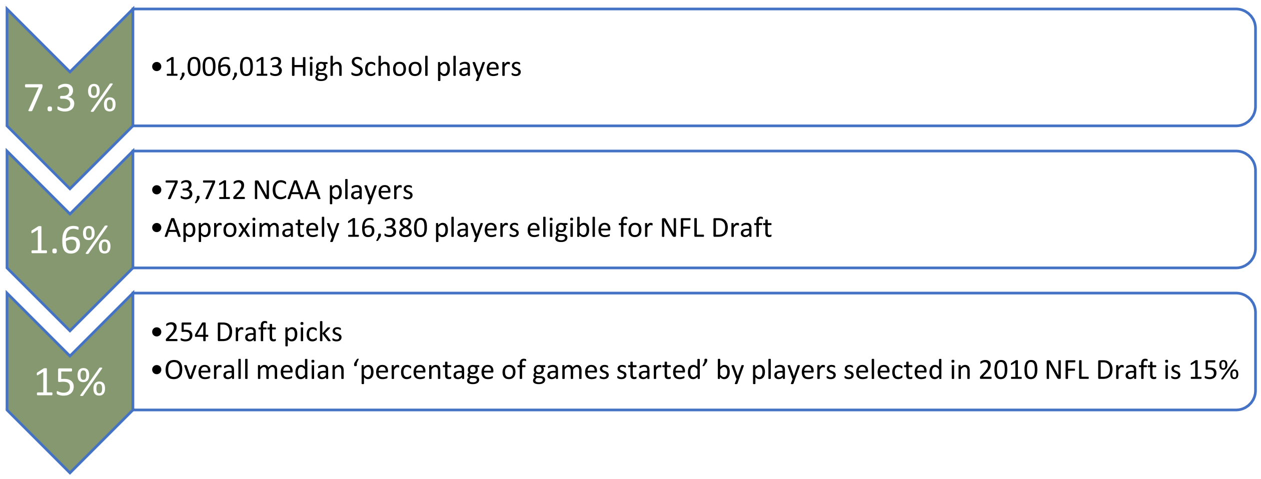
2/17/21

Final Report:

Analyzing NFL Team Statistics to Predict NFL Draft Order

Problem Statement

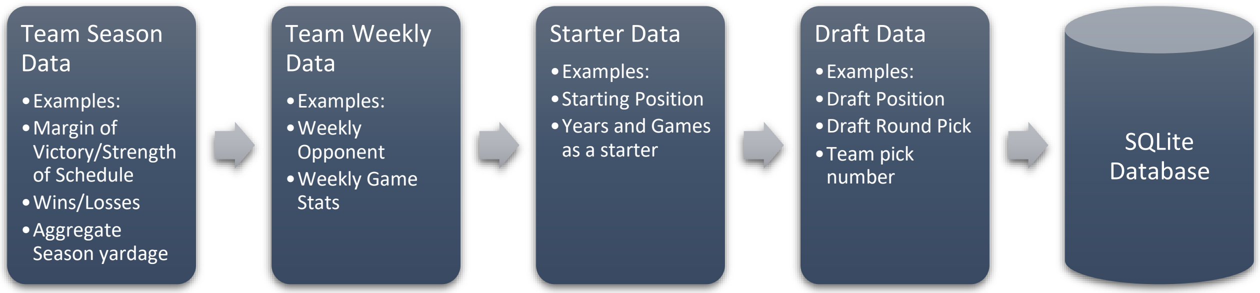
In the 2018-19 High School football season, there were 1,006,013 players across the U.S. Of the million high schoolers, 73,712 (7.3%) would eventually play as NCAA college athletes. In 2019, 16,380 of the 73 thousand athletes were eligible to enter the draft. Every April, these 16 thousand of the top football players in the country sit anxiously awaiting a call, informing them that they are one of the 255 athletes that have been drafted by one of the 32 teams in the National Football League. This represents 1.6% of the 16 thousand eligible athletes making it to the professional leagues.



- 1,006,013 High School players
- 73,712 NCAA players
- Approximately 16,380 players eligible for NFL Draft
- 254 Draft picks
- Overall median 'percentage of games started' by players selected in 2010 NFL Draft is 15%

They have been training their whole life for this moment, and rightfully so, as they look to score the two biggest wins in their sport- the Superbowl and a massive contract. In 2020 the 'rookie compensation pool' was $1.4 billion across all teams, with the first and thirty second picks getting $36 million and $10.8 million, respectively. With that much money and success on the line, it is possible to analyze historical data and determine the player position they should draft? Can predicting the draft order be determined by the team needs, without looking at what players are eligible for the draft this year?

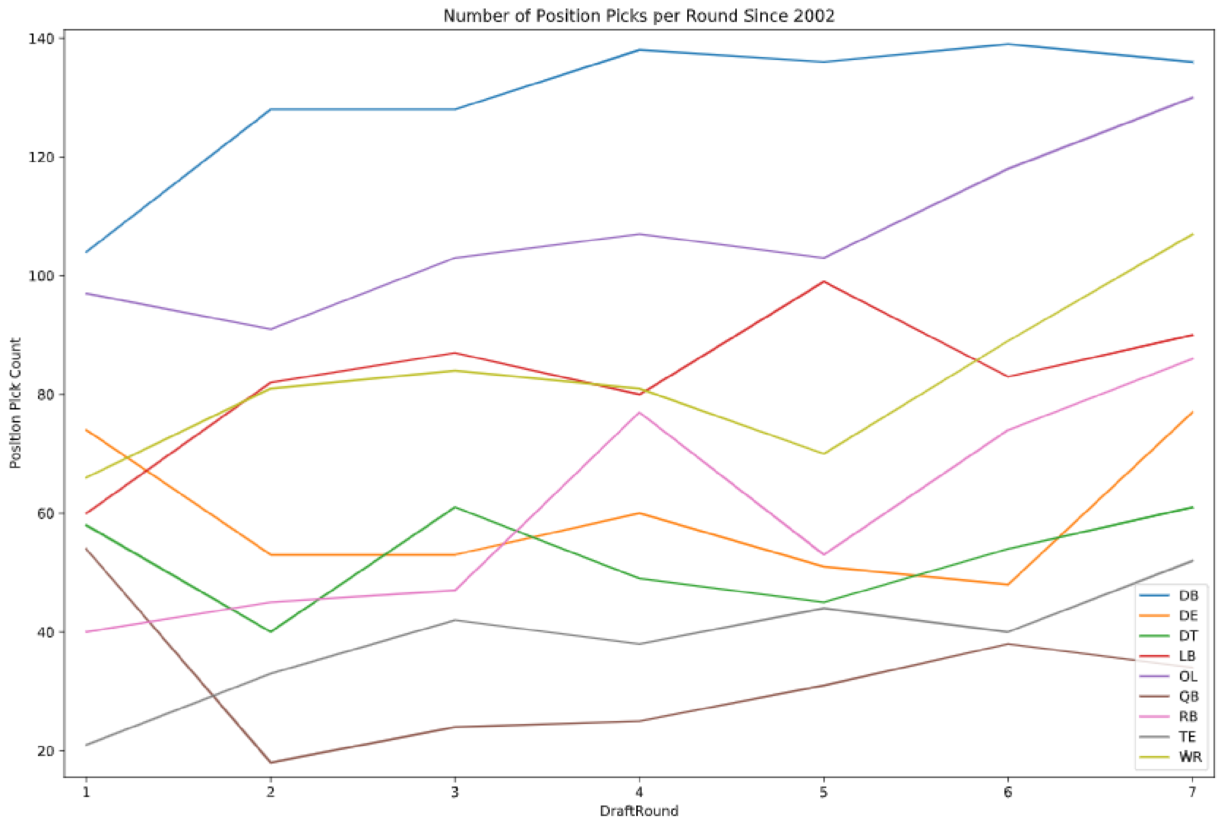
Data Wrangling



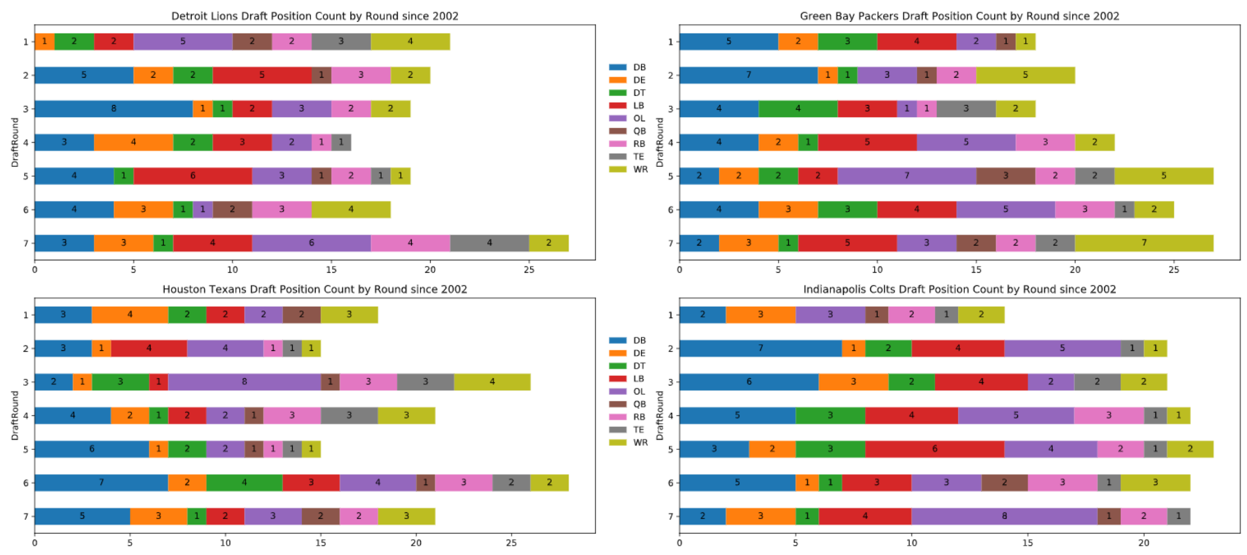
To create the dataset, all the information was web scraped data from Pro-Football-Reference.com. There were 4 sequential pages scraped for each team and year from 2002-2019, which included yearly/weekly game summaries, starting player information, and draft statistics. The years 2002-2019 were chosen because 2002 is the first year with all 32 current NFL teams (Houston Texans being added as an extension team), and 2019 being the last full season of data on Pro-Football-Reference. This information was then added to a SQLite database, to allow for more flexibility by running analysis on different query results. Each webpage scraped was added to a separate table, with a name chosen to represent the information that was scraped from each webpage. These 4 tables consist of 76 features that will be used to identify the order and player position for each draft pick. All of the data scraped was untouched, with the exception of starting player position encoding. There are many offensive and defensive schemes in the NFL, where there are different names for the same position player based on the scheme. Therefore, some positions such as Cornerback (CB), Safety(S), Free Safety (FS), had to be grouped into one position (Defensive Back (DB)) because the players are drafted for a generalized position. This allows for the reduction of potential classes for a classification model, as well as giving each position more predictive power with more instances in each class. There were only a few missing values, so the missing weekly stats were found on the internet and added to the tables.

Exploratory Data Analysis

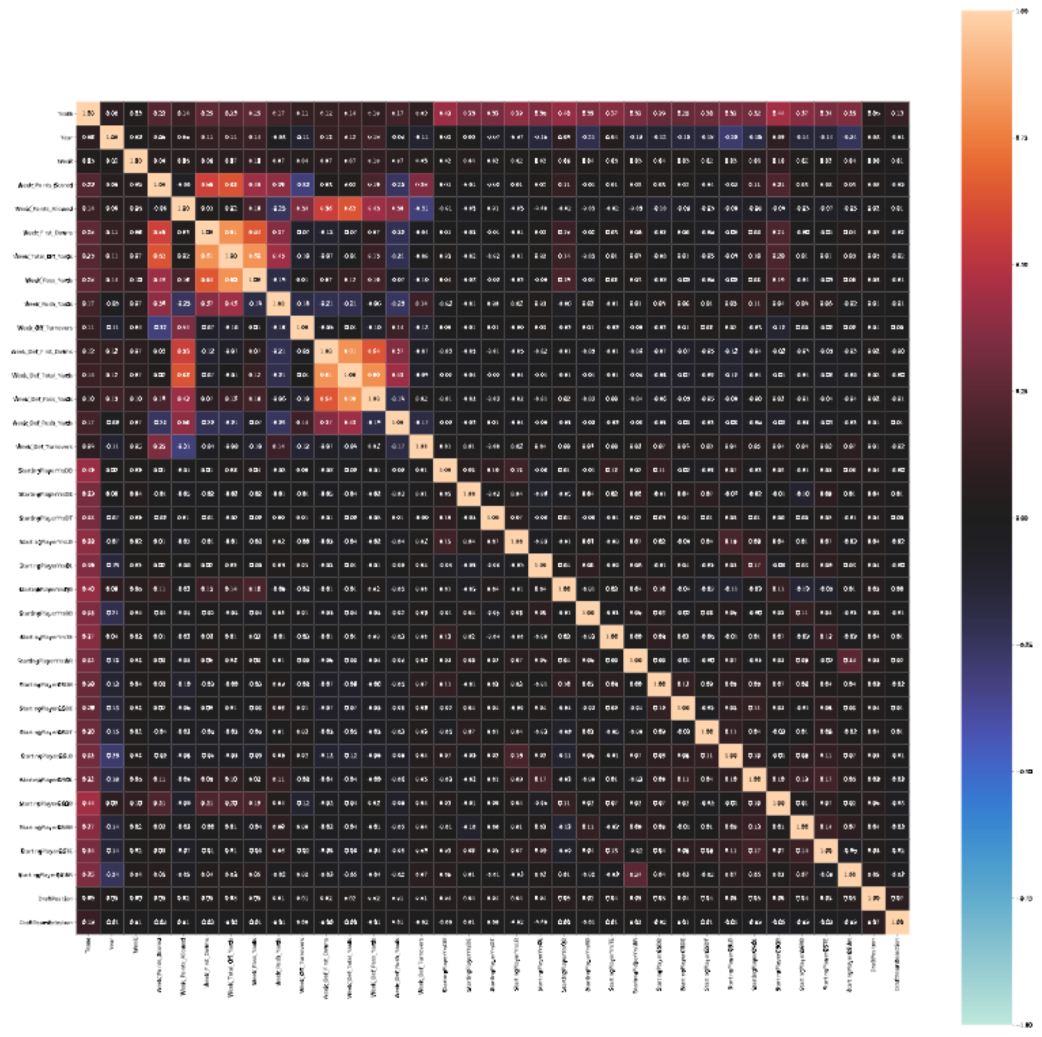
The 4 tables created by the web scraper were then turned into 4 pandas dataframes: 1) yearly summary 2) weekly summary 3) Starting Player Stats 4) Draft order. As the yearly summary was essentially an aggregation of the weekly statistics, all transformations and additional features were added to both, to compare which dataset had more predictive power. The starter and draft tables were both joined to the yearly and weekly dataframes, with the draft table containing the target features. I then found a dataset with player rankings, and joined that table on to each dataframe, creating 4 total dataframes to run through the model. There was a year table, year table with player ranking, weeks table, and weeks with player ranking table. Additional features were created to give more information about the current players on the team. One of the features was how many years the player had been on the team. A player who had been on a team should have more staying power, and less need to replace them, than a new untested player. The other feature created was aggregating the rankings of each player, into the average position ranking by each team. Because of the nature of many offensive and defensive schemes, this average player ranking allowed the feature to capture the different number of players in each position. With teams being able to trade draft picks up to the day of, the 'draft round' feature was removed and converted it into the team pick order. By doing this, each draft pick would be an ordinal feature, as opposed to a nominal feature, with multiple draft picks in the same round. For instance, in 2017, there were 37 draft day trades and this order shuffling, while record setting, is still captured by the order the team chose, instead of the arbitrary round and draft pick number. Analyses of the total draft pick counts by position showed some interesting trends. There are 130 NCAA Division 1 FBS football programs, each with their own elite players vying for a spot in the NFL. With the quarterback position being one of the most important and visual positions, it surprised me to see just how few quarterbacks have been drafted since 2002. As seen in the figure below, there was a 66% drop just from the first round to the second.



While analyzing the different trends, I created a bar chart of each team's draft breakdown since 2002. When looking at the draft selections by team, you can clearly see the impact of Hall of Fame caliber players for many years. For instance, the Indianapolis Colts had Peyton Manning as a quarterback from 1998 – 2010. As seen in the below snippet of a plot, the Colts only drafted one quarterback in the first 5 rounds in 17 years. This one pick was Andrew Luck, who was selected to replace Manning after his neck injury.



After creating the final datasets, I used a data science package called dython, to make a figure that shows correlation heatmaps with both numerical and categorical features. As seen in the figure below, there are a few features with high correlation, but none that show any with the two target variables. This was an area of concern, but the hope was that a model such as Random Forest would help find features that could predict the correct classes.



Model Preproccessing

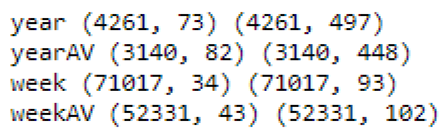
There are 4 different dataframes, each with a different number of numerical and categorical variables. Because multiple models were being used on multiple dataframes, everything was put in a pipeline within a loop. This loop would send each dataframe through the pipeline, and with multiple different models in the pipeline. Because of the nature of handling this, a ColumnTransformer was used to handle both categorical and numerical features before running the model. For categorical features, such as team name, coach, and position, the feature was one hot encoded. The categorical data needed to be one hot encoded, as there was no inherent order or ranking. Although there were no missing values in the dataframe, when mapping out a groupby into a single index dataframe, missing values popped up throughout. These values were imputed with the median values as part of the pipeline. After imputing the missing values, all numeric features were transformed using a minmax scaler. Features such as wins were single digit numbers, while yards were 4 digits, and needed to be in the same range to give equal weight to the features. The distribution of the data was normal, so there was no need to normalize the data before input into the various models. Each dataframe was then split into a 67% train and 33% test split. This data holdout prevented the models from 'peeking' at any test data, to see how well it would generalize to new, unseen data in the test split.

Model Selection

By reconfiguring the target variable, between team draft pick and position pick, this problem could be run as a regression or a classification model. Each dataframe would pass through the preprocessing pipeline, split into train/test datasets, and then tune hyperparameters using various values in a grid search 5-fold cross validation. This 5-fold cross validation takes the training dataset and splits it 5 more times. For each of the 5 new splits, this would be split again into a smaller train/test split. The score of each fold on the 'split test set' was then averaged with the 4 other folds to give an estimate of the model's predictive performance.

For the regression models, Linear, Lasso, Ridge, Elastic Net, and Random Forest Regression were chosen to try and predict the team draft pick order. Linear Regression was chosen to see how a basic model would perform. The hyperparameter tuned for Linear Regression was whether the regressors X are normalized before regression. Because there were a lot of features, as well as encoded features, regularized regression was used to penalize the extraneous features. The regularized regressors included Ridge, Lasso and Elastic Net regressors. The normalization checked in Linear Regression was also tried with the regularized regressors. In addition, for Ridge and Lasso Regression, a range of values were passed for the alpha hyperparameter, which controls the Regularization strength. For Elastic Net Regression, the L1 ratio was tuned, to vary the combination of L1 and L2 penalty. Random Forest Regression was chosen because of the large number of dimensions. For Random Forest Regression, the hyperparameters tuned included the number of estimators, the max depth, and the scoring criterion. The number of estimators represents the number of decision trees in the forest, and the max depth prevents these trees from growing to overfit the data. The scoring criteria was varied to see the impact of various scoring criteria on model selection.

The original 4 datasets were then run into each regression model, with poor results. The categorical features in the datasets were being encoded into many additional features because of their high cardinality, and the predictive power of the features was being diluted by this high cardinality(see figure below). The minimum this increased the features was by 137% in the week with player ranking dataset, to the maximum of 581% increase in the features in the year summary dataset. To remove some of the cardinality of encoded features, the categorical features were bucketized to include the values with many instances and an 'other' bucket. Bucketizing features such as head coach would allow the most prominent coaches to potentially help with classification, while removing some of the noise of lesser-known coaches. The results of these new datasets did not provide any additional accuracy of the model. Principal Component Analysis (PCA) was then used on the datasets for additional dimensional reduction. The additional benefit of PCA being applied to the model is to remove any collinearity between the numerical features. This would prevent undermining the statistical importance of the independent variables. Principal Component Analysis also allows for a selection of the number of components used by the model. This allows for selection of features that explain the majority of the variance in the data and leave out the features that do not help. Even with bucketization and PCA, results were similar between datasets, and not much improvement by any model type. To see the effect of the categorical variables and their predictive power, datasets were included that removed all categorical variables except the team name. These additional datasets did not show any additional improvement in any regression scoring metrics. This low performance may be explained by the fact that the same position can be selected in different rounds, leading to the model having problems choosing which round to pick. This led to the belief that the problem may better be solved with classification models.



```
year (4261, 73) (4261, 497)
yearAV (3140, 82) (3140, 448)
week (71017, 34) (71017, 93)
weekAV (52331, 43) (52331, 102)
```

By changing the target variable, it was time to try and solve this problem with classification models. The classification models were then run with the original datasets and included Random Forest and Support Vector Classification (SVC) models. The Random Forest Classifier was chosen because of the high dimensionality of the datasets, as well as the strength of ensemble methods. For the Random Forest model, the hyperparameters tuned in the Grid Search Cross validation were the number of estimators, the criterion function to measure the quality of the split, and the maximum depth of the tree. The number of estimators ranged from 64 to 2000 tree in the forest, the criterion functions were for Gini impurity and 'entropy' for information gain, and the max depth ranged from max depth of 1-50 and no max depth restrictions. The Support Vector Classifier was chosen to utilize the benefit of the kernel trick to deal with the high dimensionality of the data. The kernel trick would allow the original dataset to be projected into a higher dimensional space. A hyperplane would then be used to separate the data into the various classes. The kernel trick allows the operation in the original feature space without computing the coordinates of data in a higher dimensional space. The hyperparameters tuned for the Support Vector Classifier was the regularization parameter C. The C value was tuned from a value of .25 to 1, in increments of .25.

All 6 datasets were run through the two classification models, but the highest accuracy and f1 score were on the weeks dataset with 26.9% and 24%, respectively. With the highest accuracy and f1 score, the weeks dataset was chosen to be run against a dummy classifier. This was done to see how much better the best model was doing against chance. Random Forest and SVC classifiers were then run with multiple trimmed hyperparameter models against a dummy classifier using a stratified, most frequent, and uniform strategy. The Random forest classifier had the top 5 accuracy scores within the 60 model configurations attempted with GridSearch Cross validation. This also had an 115% increase in accuracy over the best dummy classifier, with 43% accuracy vs 20%. The mean test score of my 5-fold cross validation yielded a mean test score of 46%, and a model accuracy of 43% with an f1 score of 43%. An issue with this model was found when looking at the accuracy on the training set. By predicting the classes on the train set and comparing it to the actual value, the model had an accuracy of 100%. This means that with no depth limit, the trees were overfitting and not generalizing to the new, unseen data. A new model was run with depth limits in the cross-validation search, but made the model worse. This pruned random forest had a train accuracy/f1 score of 64%/65%, a test accuracy/f1 score of 35%/30%. The first unpruned model was selected, while overfitting, still generalized to the new unseen data better than the pruned forest model.

Takeaways/Further Research

Although the model did better than a chance classification, it still was correct less than half of the time. Further feature refinement can be done to increase the accuracy of classification, but it is unclear how much this would increase the accuracy. The data that was scraped may also not have the predictive power needed to complete such as prediction, so further research into more quality data may be needed to increase the predictive power needed to use this in a commercial setting. This could include adding more features to the data, such as trades and injury information. Further research into other classification would be ideal, as things like Random Forest can handle the high cardinality categorical features, without the feature predictive power being diluted by encoding.

References

https://www.pro-football-reference.com/

https://github.com/leesharpe/nfldata

https://www.forbes.com/sites/kurtbadenhausen/2020/04/24/2020-nfl-draft-first-round-rookie-salary-projections-what-burrow-tua-and-chase-young-will-make/?sh=703b5eff5be3