Capstone #1 Project Report:

Predicting a Player's NFL Draft Round

Pawandeep Jandir

Introduction

The Problem

Professional sports are a lucrative and competitive industry. The National Football League (NFL) is the biggest sports league in the world. As is the norm in American professional sports, amateur players are selected via a league-wide draft, in this case the NFL Draft. A total of 256 selections are made over 7 rounds. In the lead-up to this event, the NFL holds the NFL Scouting Combine (Combine): a long-running showcase where incoming players perform physical and mental tasks for NFL teams. Examples include the 40-yard dash, vertical jump, 3 cone drill, Wonderlic (intelligence test), and personal interviews. While this is only a portion of the draft scouting activity, it has tended to give lots of weight to the overall process. This analysis will attempt to tease out any predictive power that exists between the quantitative Combine results and draft round. Lots of resources are spent scouting players and determining draft rankings. This project can help a team provide a better draft grade for players and can also be used to estimate when a particular player may be drafted so that a draft trade can be used to target that specific player.

The Client

The potential target audience are NFL teams. They would be given a prediction model which outputs the probability of which round a player would be drafted in. They can incorporate this into their draft rankings and strategies. This can allow a team to target certain players with greater certainty given their own assigned draft picks or traded draft picks.

The Data

Three data sources are used in this project. The first is from the NFL Savant website and has data from 1999 to 2015. This is not the only website with this information but it also has provides measurements for player hands and arms. In order to check for player and drill/task completeness, Sports Reference is used. The third source, DraftHistory, is needed for NFL Draft result completeness. We know that not all players who participate in the Combine get drafted. Conversely, not all players who get drafted participate in the Combine. Further, not all players who are at the Combine will participate in all drills and tasks. So it is expected that there are null values throughout the Combine datasets. This must be kept in mind throughout the entire process.

Potential player college data and statistics can also be used to perhaps add more predictive power to this analysis. Not all colleges produce equal amounts of draft picks. Some colleges are known powerhouses, churning out many players who end up being drafted, often quite high. Additionally, being on a good or high performing college team also boasts a players profile and chances in the NFL Draft. So adding this type of information could provide additional handles on predicting draft round.

The Tools

The entirety of this project uses *python* and *JupyterNotebook*. Some of the python packages used are *numpy*, *pandas*, *BeautifulSoup*, *matplotlib*, *seaborn*, *scipy*, *statsmodels*, and *scikit-learn*. The entire project can be found on github here. The code and scripts can be run completely out of the box, as the repository includes all related code and data. Reports, such as this one, can also be found in this online repo for further details and discussions.

Data Wrangling

Acquisition

After identification of data sources, the next step is to obtain it somehow. The list below gives the details for each data source.

1. NFL Savant : This is by far the easiest of the trio. The site offers a csv file of the Combine data. This can then be read into a pandas dataframe very easily.

2. Sports Reference : The data stored by Sports Reference is in html tables. Thus, the site has to be scraped using *requests* and *BeautifulSoup*. After getting the full contents of the html table using those packages, we construct a function which can create a *pandas* dataframe from it. After this dataframe is created, a single column consisting of draft team, round, pick, and year is split up into their own separate columns with `str.split()`. Each year is on a separate page, so we loop through all the webpages one at a time while appending each created data frame. At the end, the dataframes are concatenated to a single dataframe. To avoid making repeated calls to the Sports Reference website, this dataframe is saved (via *pickle*) locally. This should allow for better reproducibility as well.

3. DraftHistory : Similar to Sports Reference, Draft History also has their data available on their website in html tables. A very similar approach was used since much of the framework to scrape was already in place. The approach is identical up to the html table to *pandas* dataframe conversion. After that, there are differences in how the resultant dataframe should be formatted. The draft round column is littered with "\xa0" strings so that needs to be removed. Also the draft round column needs to be forward filled since the website html tables only displays the draft round number when there was a new round. Looping over the relevant years, this dataframe is also saved to a local pickle file for later analysis.

Cleaning

The time needed to clean the data and ensure data quality is large, so let's get started! The three datasets are worked in order. Immediately an issue is spotted when we read in the NFL Savant csv file. Two rows had elements which had commas in the field. Obviously this is a problem for csv files. However, the issue was confined only to two rows, so instead of trying to find a general method to deal with such edge cases, we can just manually chang the csv file to remove the offending commas. This way, we can load up the data and continue, quickly.

Considerable time was spent looking at what data was missing and how best to fill it (if at all). One way already mentioned was using additional datasets. We load up the Sports Reference dataframe and ensure the columns are the correct type. However, it is obvious this dataset is also not complete. We use `pd.merge()` to full outer join the two dataframes and include an indicator variable for manual inspection of the joined dataframe. We wrote a small function which compares and combines the values of two columns (one from each dataset) to see the percentage of null values. This way there is a quantitative way to determine how combing the two datasets (for a particular column) can recover missing data. On average, there is approximately a 10% recovery. This works about how we expected. Unnecessary columns are now dropped.

At this point it is worth cleaning up the player college values. Because we joined two different data sources not every college is listed with the same name. For instance, the University of Southern California can be referred to as Southern California or just USC. This needs be consistent across whatever datasources we use. After this step, we can turn our focus on the draft round and pick data which seems to be missing more values than expected. A closer look reveals the 2014 year is totally missing these values. Even worse, both data sources mislabel draft round completely. In both they refer to the pick within a round instead of the round itself, while the pick refers to the overall pick and not the pick within a round. This necessitates inclusion of the third datasource, DraftHistory, to get the correct draft round and pick data.

This time we use a left join to merge the two dataframes. This is because not all drafted players participate in the Combine. Any players missing in the DraftHistory dataset mean they were not drafted, and thus have legitimate null values in their draft pick data. This third dataset is also used to fill the draft team column which seemed to be missing some values. After combining columns as before, a more complete dataframe is now forming.

Among the many columns in the datasets we have combined, only one was added. We grouped similar player positions together into a positiongroups column. This is because positions can be fluid going into the NFL so it is very useful to view college players at this coarser level. Additional columns may also be added relating to a player's college stats and a player's college itself in the future.

The last step is to fill in the null data throughout the dataset. There are various ways to do this imputation but we first explore the outliers and statistics in each column to better understand them. There are no significant differences in mean or median for any one column. However, there are more outliers than expected, so we should be mindful of this. An example calculation is shown below for the broad feature. The minimum and maximum values are shown alongside their respective z-score.

```
    broad| mean: 113.177 median: 114.00 std: 9.398  size: 4696
         | min:   74.000 zs_min: 4.17   max: 147.00 zs_max: 3.60
_________| observed outliers: 17        expected outliers: 12.7
```

We will then impute the null values per column. But first we will group by positiongroup first, since this will ensure a grouping of more similar players. This should mean their attributes and tests are likelier to be related than using all players in the dataset. This makes sense because we generally expect two 300 pound linemen to be more similar than a 300 pound lineman and a180 pound running back. Because we saw there are more outliers than expected, we will impute using the median of the columns, grouped by positiongroup, instead of the mean. This should be the safer option of the two.

The outliers, perceived or otherwise, will not be changed at this time since these are real measurements. The last missing values to change are the draft round and pick data. For now, those values are set to -1, though this may change later.

Features

At the end of this process we have many variables or features in our dataset. The table below (Tab. 1) lists all of these features with descriptions and their potential in the prediction model. Note the twelve variables marked with an asterisk, *, are defined as the variables of interest which will be discussed later in this document.

Table 1: Full list of features in the dataset

Feature	Description	Notes	Useful in prediction?
name	Name of the player	-	No
year	Draft year	-	Possibly
college	Player's college attended	-	Possibly
position	Player's college position	Scraped value	Yes
positiongroup	Player's college position group	Assigned value	Yes
height*	Player's height (inches)	-	Yes
weight*	Player's weight (pounds)	-	Yes
fortyyd*	Player's 40-yard dash (seconds)	Full description	Yes
vertical*	Player's standing vertical jump (inches)	Full description	Yes
bench*	Player's bench press of 225 pounds (reps)	Full description	Yes
threecone*	Player's 3 cone drill (seconds)	Full description	Yes
shuttle*	Player's shuttle run (seconds)	Full description	Yes
broad*	Player's standing long jump (inches)	Full description	Yes
wonderlic*	Player's Wonderlic test result	Sparsely populated	Possibly
nflgrade*	Player's NFL Grade determined by experts	Sparsely populated	Possibly
arms*	Player's arm length (inches)	-	Yes
hands*	Player's hand length (inches)	-	Yes
team	Team that drafts player	-	Possibly
round	Player's draft round	Target variable	N/A
pick	Player's draft pick number within a round	-	No
overall	Player's overall draft number	-	No

The full code detailing these data wrangling steps can be found on my github. A link to this $JupyterNotebook$ is given here.

Data Exploration

Initial and exploratory data analysis is performed using various visuals and inferential statistics. The purpose is to gain some insight into how draft round is influenced by all the other variables in the cleaned dataset. Because quarterbacks are generally thought of as the most important player on the team, they are given outsized importance in the draft. For that reason, we focus on quarterbacks here.

Visual Analysis

One of the first things to ask is how many quarterbacks are in the dataset per year. Is it pretty consistent over the years or does it vary? We can plot this, as shown in Fig. 1. Each year has the number of quarterbacks printed at each point. The mean is represented by the red dotted line.

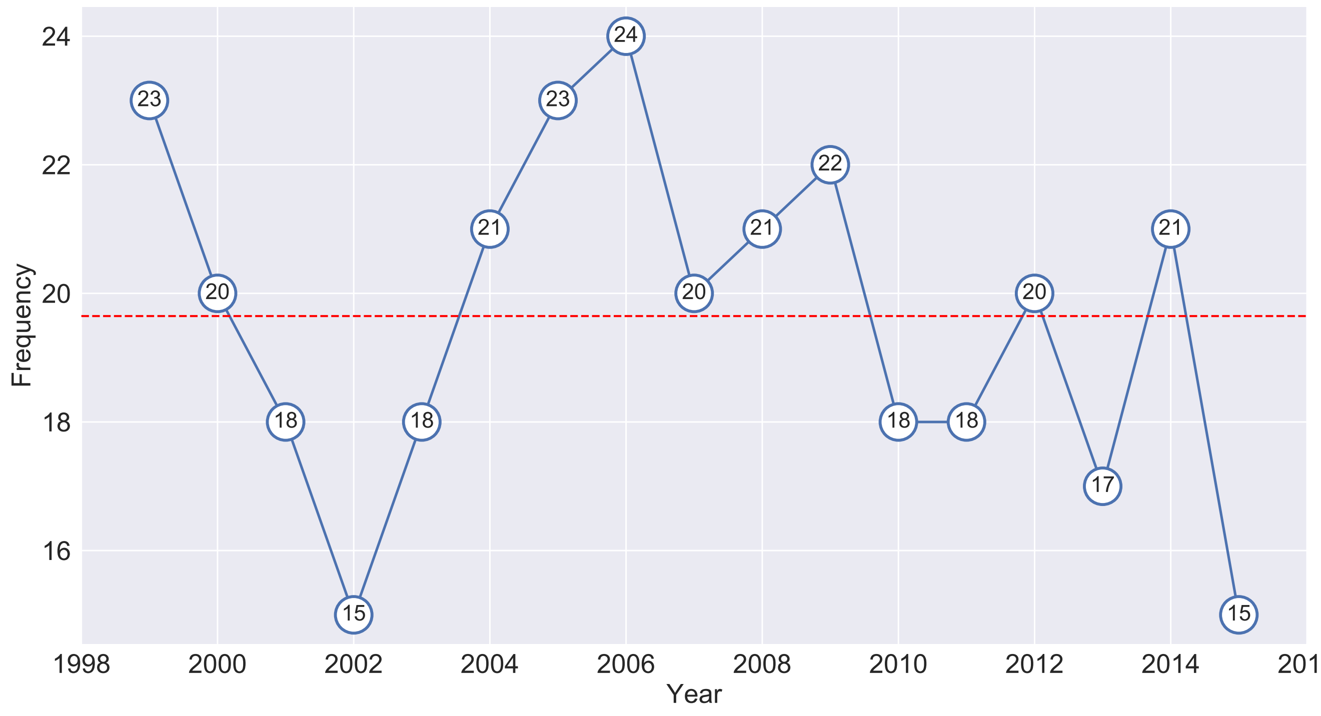


Figure 1: Quarterbacks per year.

We can see that generally there are 20 or so quarterbacks every year, with some notable exceptions. Along with a few years with an above average number of quarterbacks, like 2006, there are a few years with a below average number of quarterbacks, like 2002. There are probably a multitude of factors which we can hazard a guess for, but it is likely outside the scope of this project. We can also look at the distribution of quarterbacks as a function of draft round. Remember that a round of -1 means the player was undrafted. Shown below, in Tab. 2, is the table of quarterback count per round.

Table 2: Quarterbacks drafted per round

Draft Round	-1	1	2	3	4	5	6	7
Count	141	47	19	22	23	29	27	26

It should not be too surprising to see that a large portion of quarterbacks do not get drafted. The quarterback is the most important position on the team so it is also the most scrutinized. There are only a few quarterbacks on any single team, so teams are judicious in selecting potential replacements. In that same vein, it should be expected that the first round is the most popular to draft a quarterback. It is the first opportunity to select a player and quarterback-needy teams likely need to prioritize that position over others. This importance means teams often gamble to get “their” guy who can be the starting quarterback for a decade or more. It is is also worth checking out how quarterbacks are drafted per round per year. This is best visualized in a heatmap, as shown in Fig. 2 below.

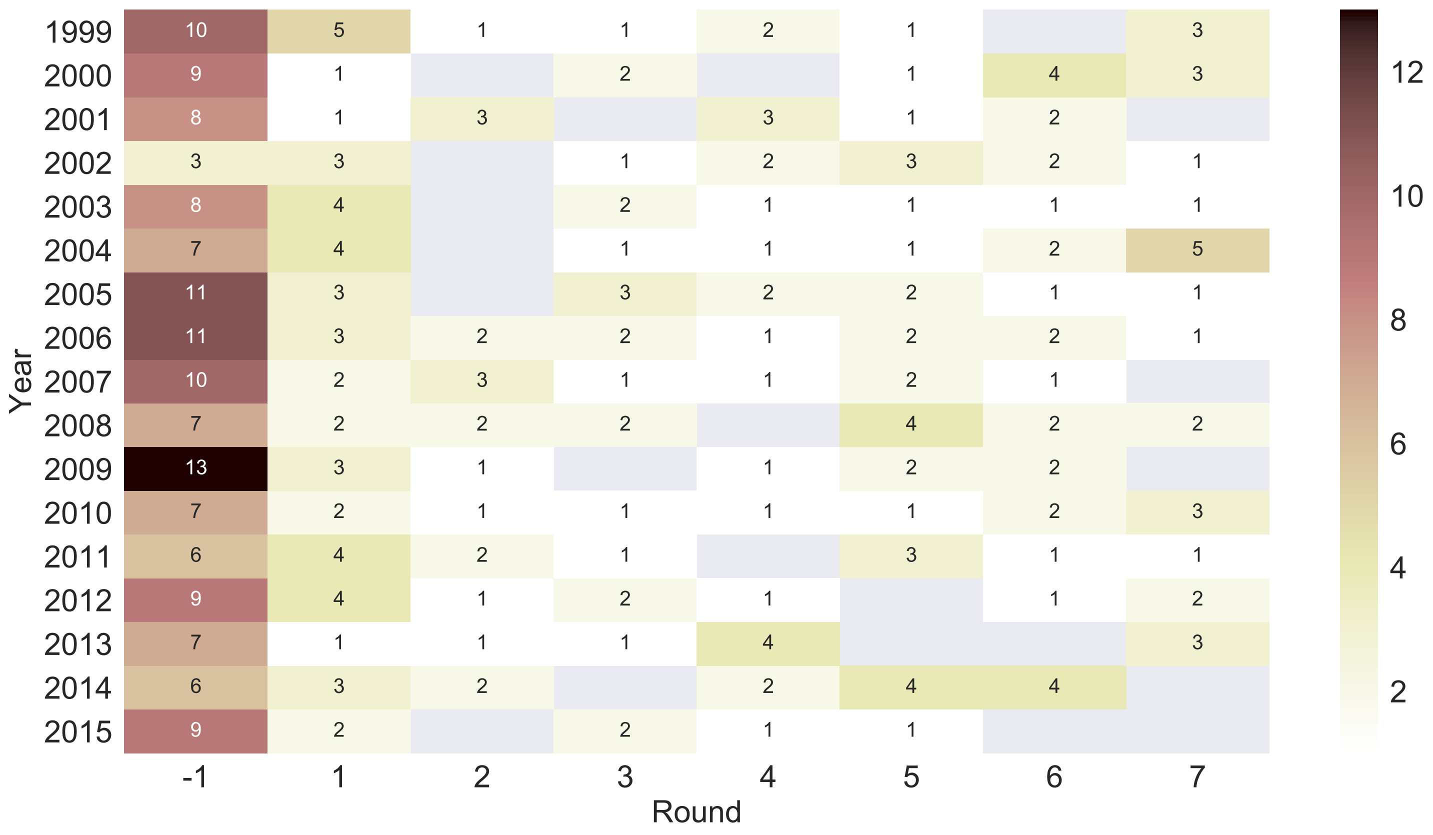


Figure 2: Quarterbacks per round and year.

We can look at this plot along with the previous plot to note some interesting years. In general, a quarterback is always selected in the first round. Again, no surprises here. The 2009 draft is quite surprising. It had an above average 22 quarterbacks but also let 13 of them go undrafted. Yikes. Only 2015 had a (marginally) worse percentage of players go undrafted.

We can also look at which teams have drafted quarterbacks by plotting it. This is done in Fig. 3 below. The red line represents the median of the number of quarterbacks drafted per team.

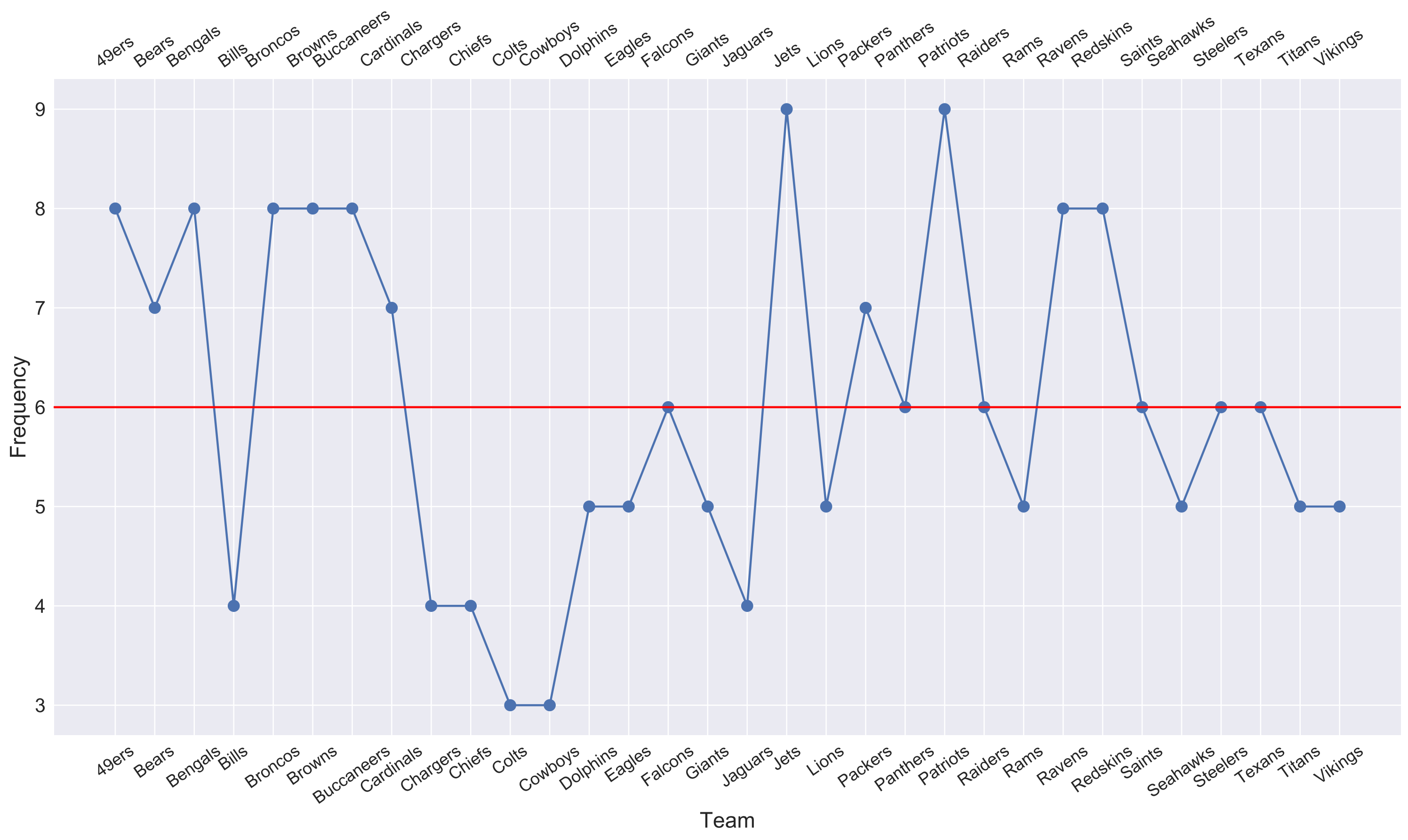


Figure 3: Quarterbacks drafted per team.

Considering this dataset covers 1999 to 2015, the teams with the most drafted quarterbacks make some sense (given knowledge of the NFL landscape). The teams who have tended not to have a long term starter at quarterback during this period show peaks in this distribution, although a notable exception does exist with the Patriots.

Let's switch gears to some of the actual measurables in the data along with the draft round information (the target variable in the analysis). We can pick out those interesting columns, as mentioned in Tab. 1 earlier, and make a heatmap as in Fig. 4. This shows an overall picture of how the variables are correlated. Positive and negative values are in purple and green, respectively.

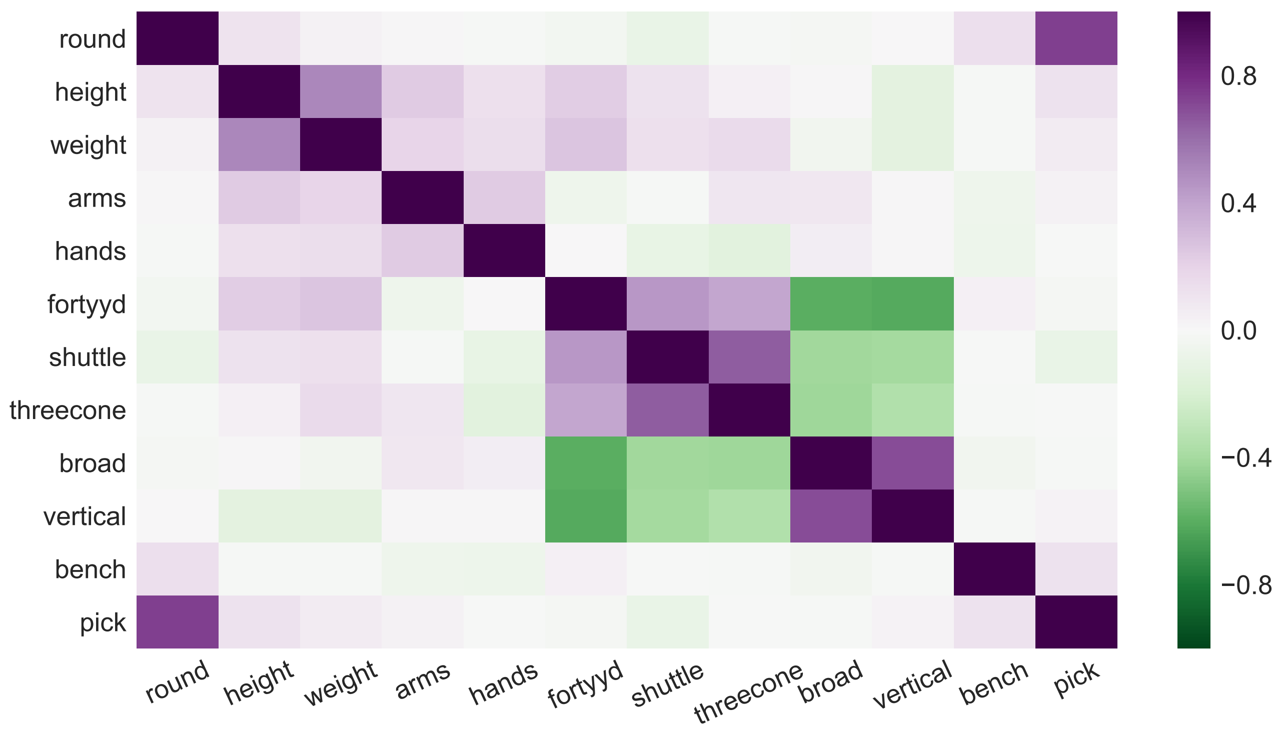
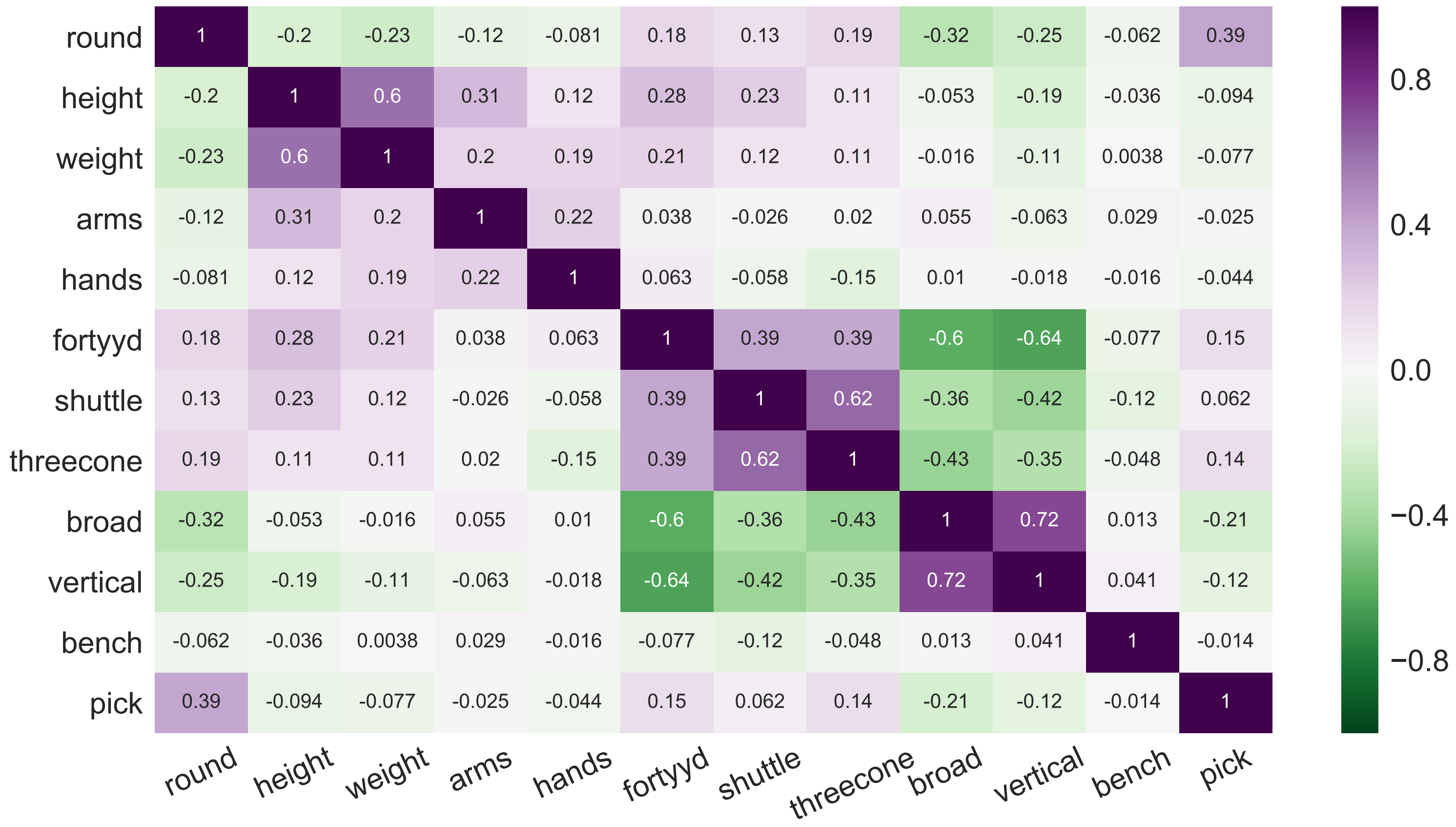


Figure 4: Correlations for all quarterbacks.

We can see how certain variables are pretty well (anti-)correlated. For instance, the fortyyd and vertical are relatively strongly anti-correlated. These values represent the 40 yd sprint and vertical jump drills, so perhaps this should be expected: the muscle groups involved are different for each drill. This is reinforced by the strong correlation between the broad (or standing long jump) and vertical jumps. However, this data is for all quarterbacks in this dataset. How does it look for quarterbacks that were actually drafted? We can re-do this heatmap as in Fig. 5 with that constraint. As an added visual aid for this plot, we can also print the correlation values.



	round	height	weight	arms	hands	fortyyd	shuttle	threecone	broad	vertical	bench	pick
round	1	-0.2	-0.23	-0.12	-0.081	0.18	0.13	0.19	-0.32	-0.25	-0.062	0.39
height	-0.2	1	0.6	0.31	0.12	0.28	0.23	0.11	-0.053	-0.19	-0.036	-0.094
weight	-0.23	0.6	1	0.2	0.19	0.21	0.12	0.11	-0.016	-0.11	0.0038	-0.077
arms	-0.12	0.31	0.2	1	0.22	0.038	-0.026	0.02	0.055	-0.063	0.029	-0.025
hands	-0.081	0.12	0.19	0.22	1	0.063	-0.058	-0.15	0.01	-0.018	-0.016	-0.044
fortyyd	0.18	0.28	0.21	0.038	0.063	1	0.39	0.39	-0.6	-0.64	-0.077	0.15
shuttle	0.13	0.23	0.12	-0.026	-0.058	0.39	1	0.62	-0.36	-0.42	-0.12	0.062
threecone	0.19	0.11	0.11	0.02	-0.15	0.39	0.62	1	-0.43	-0.35	-0.048	0.14
broad	-0.32	-0.053	-0.016	0.055	0.01	-0.6	-0.36	-0.43	1	0.72	0.013	-0.21
vertical	-0.25	-0.19	-0.11	-0.063	-0.018	-0.64	-0.42	-0.35	0.72	1	0.041	-0.12
bench	-0.062	-0.036	0.0038	0.029	-0.016	-0.077	-0.12	-0.048	0.013	0.041	1	-0.014
pick	0.39	-0.094	-0.077	-0.025	-0.044	0.15	0.062	0.14	-0.21	-0.12	-0.014	1

Figure 5: Correlations for drafted quarterbacks.

The correlations between these variables seem to rise when we require the quarterbacks to have been drafted. However, is this a real effect or just a coincidence since we reduced the number of players by 42%? It might be a little of both and perhaps we can explore this effect at a later time. We also must remember another limiting factor in this type of visualization: draft round should probably be a categorical variable. As such, calculating the correlation between draft round and another (numerical) variable is not so easy. Regardless, as long as we are aware of this fact we can still derive some useful meaning.

At the end of the day however, our target variable is the draft round. So it would be good to see correlations in that row (again with the limitations in mind). We can choose a few to look into greater detail by generating a "joint plot" to visualize these bivariate distributions. Let's take closer look at how the broad jump and draft round data are related for quarterbacks.

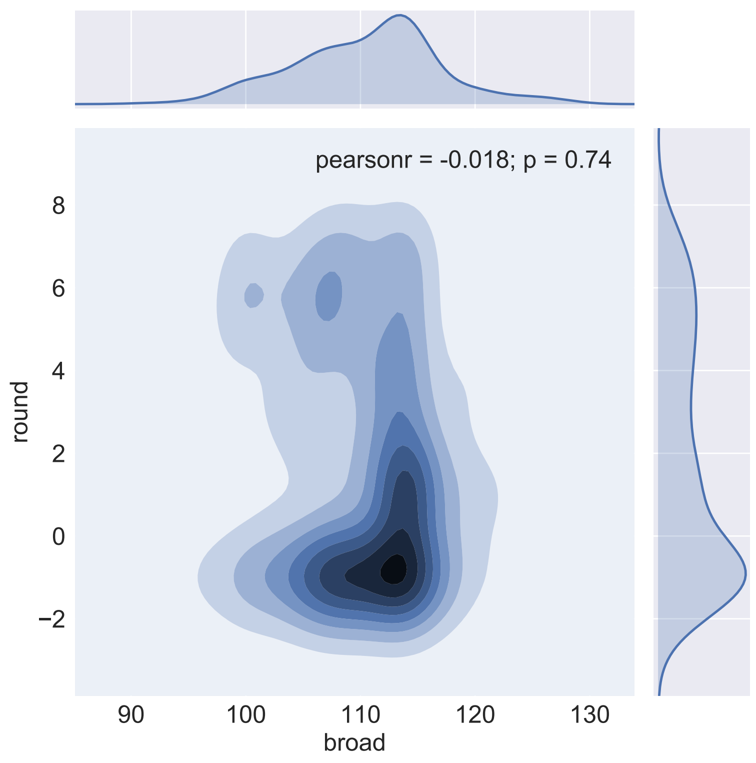


Figure 6: Draft round vs broad jump for all quarterbacks.

A kernel density estimation (kde) is used to help with the visuals of this type of plot. We can see how the concentration of undrafted quarterbacks affects this relationship: the clustering of undrafted quarterbacks has a large effect and a lot of the undrafted quarterbacks have a similar broad jump. A lot of the undrafted quarterbacks have a similar broad jump, with the imputation perhaps playing a large role. Since we want to predict the draft round variable, correlations like these are necessary. However, the draft round is a categorical variable, even if it ranges from -1 to 7 (no zero). This means the usual correlation calculation, the Pearson correlation coefficient, is not truly appropriate to use this case. Instead we can use the Kendall rank correlation coefficient to determine correlation. This test measures the ordinal association and is better suited for our purposes than the Pearson correlation coefficient.

The full code detailing this visual data exploration can be found on my github. A link to this $JupyterNotebook$ is given here.

Statistical Inference

As a statistical test, the (Kendall) correlation also gives us a p-value. The null hypothesis in this case is that there is no correlation between the two variables being tested. So a low p-value suggests the calculated correlation is significant. We can assume the usual statistical significance level of 5%. However, in order to really trust the result of this test we can perform a permutation test of the correlation. This permutation test randomly exchanges the draft round labels and re-calculates the correlation. This permuted correlation is calculated many times to develop a histogram (see Fig. 7). From this we can determine a permuted, or estimated, p-value by dividing the number of permuted correlations greater than the observed correlation by the total number of permutations done. This estimated p-value can then be compared to the observed p-value to determine which correlations are significant and which ones are not.

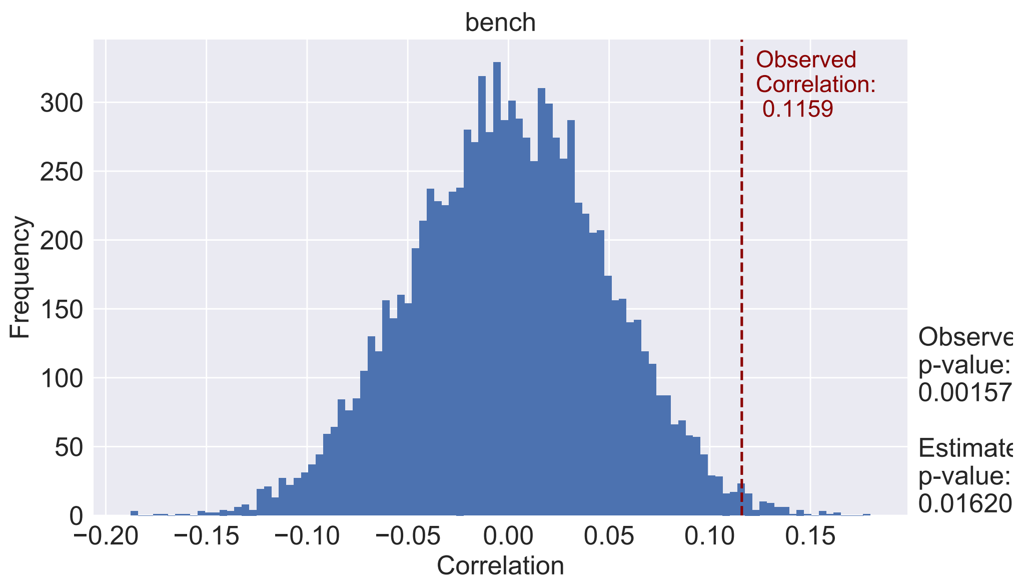


Figure 7: Permuted correlations between draft round and bench press.

After performing this test across the twelve variables of interest, only the height, bench and nflgrade columns show significant correlations to draft round. The bench correlation test is shown in Fig. 7. However, it must be noted that the highest measured correlation is about 0.12, which is quite weak. Regardless, even if they are weak correlations, they still seem to be real, significant ones.

The previous test focused on all of the quarterbacks. What if we separate the quarterbacks who were drafted from those who went undrafted? Are there differences in these types of players? For this test, we can measure the difference in means of each variable of interest. Then we can calculate a z-test and its associated p-value to get the statistical significance. The null hypothesis in this test is that the difference in means between the drafted and undrafted quarterbacks is zero. A small p-value (i.e. $\alpha = 0.05$) implies that we can reject the null hypothesis. When performing this test, we determine that in most cases the p-value is smaller than 5%. There are only three variables with a p-value larger than 0.05: threecone, arms, and hands. Therefore, for these columns, we cannot reject the null hypothesis. However, in all others, we likely can. This means in general, there is a statistical difference between quarterbacks that are drafted and those that are not drafted. An example variable of interest, the shuttle (run) is shown below.

```
   shuttle | z-test statistic :-3.125969
           | z-test p-value   : 0.001772
           | 95% Conf Int     :-0.088894 to -0.020380
-----------|
```

We can expand our focus beyond just quarterbacks and include all players as well. By dividing players into their positiongroups, we can compare how the distribution of a variable of interest is different in any two positiongroups. We can again use a z-test to get the significance in the difference in means of the distributions. For example, we can compare how a defensive lineman's (DL) bench press distribution agrees with a linebacker's (LB) bench press distribution. A z-test and its p-value can help us determine whether there is a statistical difference or not. We can go further and loop through all combinations of positiongroups for each variable of interest. The result is a heatmap of the p-values of the z-tests mapping each of these possibilities. In this type of plot, a low (light) value implies statistically signficant, while a high (dark) value implies no significance. The null hypothesis here is that the difference in means is zero. Remember each variable of interest generates a separate heatmap. The max color scale value is capped at 0.1 to ensure all dark colors land outside the α level we have set (at 5%). This heatmap is also symmetric by construction. An example of such a heatmap is given in Fig. 8.

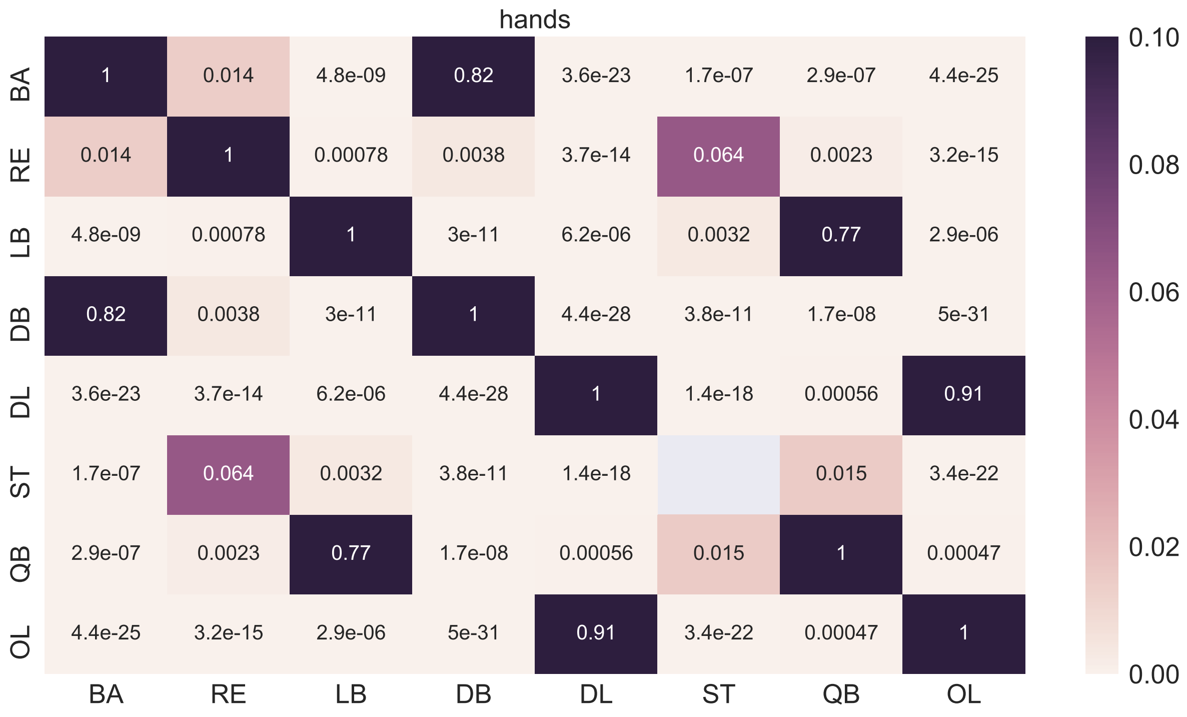


Figure 8: z-test p-values for hands variable for all positiongroups.

From this example, we can see how pairs of positiongroups interact with each other for the hands variable. Offensive linemen and defensive linemen share a very similar mean as it has a p-value of 0.91. However the means are very different for offensive linemen and quarterbacks as the p-value is 0.00047, well below our 5% significance threshold.

The full code detailing this visual data exploration can be found on my github. A link to this $JupyterNotebook$ is given here.

Data Modeling

Preprocessing

Having analyzed the dataset, we move towards building a suitable model to predict the round where a player is drafted. One of the first things to do at this stage is to convert categorical variables to dummy ones (i.e. one-hot encoding). We only have one categorical variable that we are keeping at this stage, positiongroups. Note also the first dummy variable is dropped since we can infer a positive value (i.e. 1) for a player when all other positiongroup dummy columns are a negative value (i.e. 0). In this way, we end up with 20 features, or columns, to use in our modeling. These features are shown in Tab. 3 with their respective value type as well.

Table 3: List of features in model building

Feature	Type
year	Integer
height	Integer
weight	Float
fortyyd	Float
vertical	Float
bench	Float
threecone	Float
shuttle	Float
broad	Float
wonderlic	Integer
nflgrade	Float
arms	Float
hands	Float
positiongroup_DB	Dummy (binary)
positiongroup_DL	Dummy (binary)
positiongroup_LB	Dummy (binary)
positiongroup_OL	Dummy (binary)
positiongroup_QB	Dummy (binary)
positiongroup_RE	Dummy (binary)
positiongroup_ST	Dummy (binary)

Data standardization

Some features in our data have very different ranges than other features. The year column has values from 1999 to 2015, while the fortyyd column has values from 4.2 to 6.1. These disparate values can mislead a classifier as generally only the relative changes within the column have value not the raw changes. In this type of dataset, features should normalized in some way to put them on equal footing. Various methods exist to do this. In the case of the simplest model we will use, see Logistic Regression below, a `MaxAbsScaler` from *scikit-learn* is used. This scales each feature by its maximum value so that it ranges from 0 to 1. Other methods include `StandardScaler` which centers and scales the variance to one for each feature.

Dimensionality reduction

We can explore some dimension reduction with this data. With the dummy variables, we have 20 features in our dataset, so It might be prudent to try to see if reducing the dimensionality of the dataset is useful. We can do this by first scaling and normalizing the data, as discussed above, and applying Principal Component Analysis (PCA). Then we can visualize the results of the analysis to see if it helps. Fig. 9 shows this result.

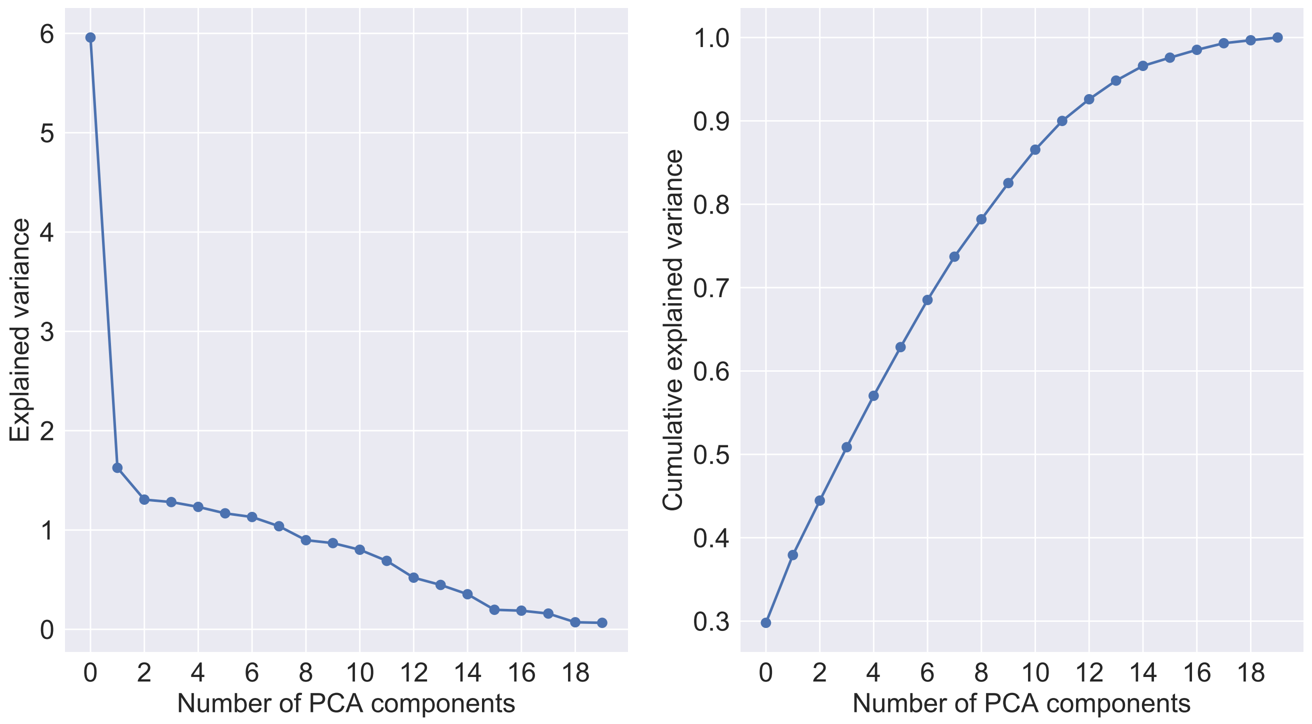


Figure 9: Test of dimensionality reduction

By plotting the explained variances for various number of PCA components, we can see if there is a good dimensionality that exists which explains most of the data but in fewer dimensions. Usually we want the cumulative plot (right plot) to show an elbow or kink indicating an optimal number of PCA components. This plot has no clear suggestion which is corroborated by the right plot. A value between 2 to 6 components might be the best as higher values show diminishing returns.

While not discussed in this document, through testing its been seen that no number of PCA components seems to help. So while a useful exercise, PCA did not improve any model by any significant amount. Therefore, all tested classifiers were fed all features in the dataset with no reduction.

Cross-Validation and Scoring

We now have the full dataset we want to model. Now we determine how we want to score and measure a classifier. That is, what metric will we use to decide which model is good and which is not? Naively we can use accuracy as a metric. Recall accuracy is the fraction of correct predictions. For binary classification problems this can work well. However, for multi-class predictions, like this one, it can be an exceptionally harsh metric. Instead we will turn to a (weighted) F1 score. The F1 score is a harmonic mean of precision and recall: $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. Precision is defined as the ratio of true positives divided by the total number of predicted positives (also known as positive predictive rate). Recall, also known as the true positive rate or sensitivity, is defined as the ratio of true positives divided by the total number of positives. In the case of a multi-class problem these definitions are modified slightly to average over the individual classes. In other words the performance of the classifier is averaged per class. So the precision, PRE, would generalize to:

$$\text{PRE}_{\text{Multi}} = \frac{1}{k}\sum_{i=1}^{k} \text{PRE}_i \cdot \text{Weight}_i,$$

where Weight_i is a factor inversely proportional to the class frequencies. The sum is over the k number of classes in the problem, which in our case is 8. Similarly the recall, REC, would generalize to:

$$\text{REC}_{\text{Multi}} = \frac{1}{k}\sum_{i=1}^{k} \text{REC}_i \cdot \text{Weight}_i.$$

Thus, the F1 score we will use to discriminate between models uses these definitions of precision and recall. Another aspect of differentiating between models is to use cross-validation. This strategy takes a few different forms in this analysis. First, we divide up our data into two pieces: a training set (80%) and a testing set (20%). We will use only the training data to fit our models. Then they will be evaluated against the testing data to determine their performance. Second, in order to optimize the algorithms we will use a cross-validated grid search. This will be discussed in further detail below as we go through testing a model. Lastly, the test data itself will also be used in a cross-validated method. This test set is further divided into sub train and test sets. The models are fit on this sub-training data and scored on the sub-testing data. The method of both cross-validation strategies is a (stratified) k-fold scheme. Data is split into k-blocks and the first block is held out as a test set and the rest as a train set. The model is again scored against the test set. This number is saved and we then hold out the second block as a test set and the rest as a train set. This process is repeated over all k number of folds and then averaged. This strategy aims to decrease the variance of the model and get a fair estimate of its performance. In our case, we will use 5 folds (i.e. k = 5). We also use the stratified k-fold method, and not the usual k-fold, to mirror the fact that we have unbalanced classes. This can be shown in Tab. 4. This stratified method will ensure that the random sampling reflects the differences in class proportions.

Table 4: Class Proportions

Round	Proportion [%]
-1	36.3
1	9.26
2	9.26
3	9.58
4	10.2
5	9.03
6	8.03
7	8.32

Baseline

Before we start building models, we need some baseline model. We need some method to establish that our model is better than what we can do naively. In this case, since we have a class encompassing over a third of our dataset, the smartest naive thing is to always predict that a player goes undrafted. If we used this dummy classifier, we would be correct 36% of the time (in terms of accuracy). The F1 score for this type of model is about 19%. So the score to beat is that 19%. Any classifier that we build should have a higher score than this one–otherwise it is fairly worthless.

Model building

In total we will test and optimize 6 classifiers to best our baseline classifier. We will start with one of the simplest classifiers, a Logistic Regression.

Logistic Regression

As discussed earlier, we wish to optimize, on the F1 score, a Logistic Regression model. This is done via a cross-validated grid search. A (hyper)parameter grid is setup so that at each point along this grid we perform cross-validation to determine its score. After traversing the entire grid, the model with the best score is determined to be the optimal model. In the case of a Logistic Regression model, we search over only one parameter, C: the inverse of the regularization parameter. Smaller values mean stronger regularization. Using this method, we can determine which parameters work best for our data. This optimal model is then fit and scored against the testing data. All other classifiers considered here go through the same treatment.

A benefit of Logistic Regression is looking directly at the generated fit coefficients. This tells us exactly how the model has learned the data in a straightforward way and makes interpreting the results of the model much easier. This particular dataset has the additional complication of having 8 classes. Thus, there are 8 classes $\cdot$ 20 features = 160 coefficients. We can visualize this in a heatmap, as in Fig. 10. This plot is a 2D map showing the fit coefficient for each feature (y-axis) vs draft round (x-axis).

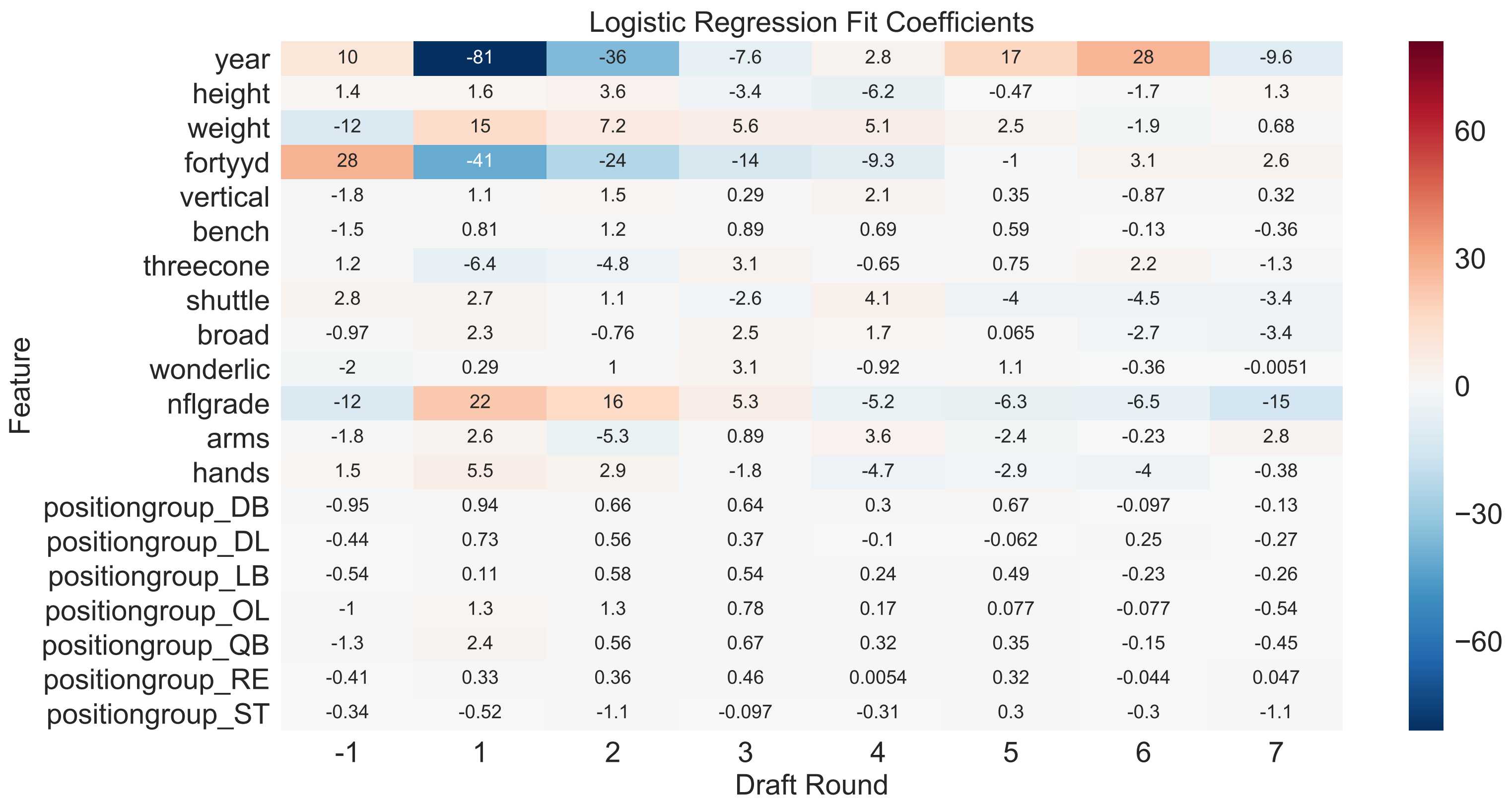


Figure 10: Logistic Regression fit coefficients

We can see that the positiongroup dummy variables do not seem to have a large effect on draft round. Focusing on the fortyyd feature (recall it is the 40yd sprint), the fit seems to say that the first few rounds preferentially have lower fortyyd times than the last few. Players who are not drafted tend to be the slowest. This seems to make sense. Similar interpretations can be made for the other features as well.

Support Vector Machine

A support vector machine (SVM) classifier is also optimized and trained in the way outlined previously. In this case, two parameters are traversed: C and γ. This parameter C is essentially a regularization term, where a small C value tends to have a weaker regularization, as a large C value tries to classify all given examples correctly. γ specifies the influence of any single given example. It should be noted that a `StandardScaler` is used for this model, as it seems to work best.

We can plot the results of the parameter search and check the performance at each point. The plot below, Fig. 11, shows the F1 score against the C parameter on the x-axis. The γ parameter is shown in different colors while the internal test and train data is shown in different line styles. Remember, this grid search uses cross-validation on the given training data to create sub-train and sub-test data. These latter two sets of data are the plotted quantities, testing data in solid lines and training data in dashed lines.

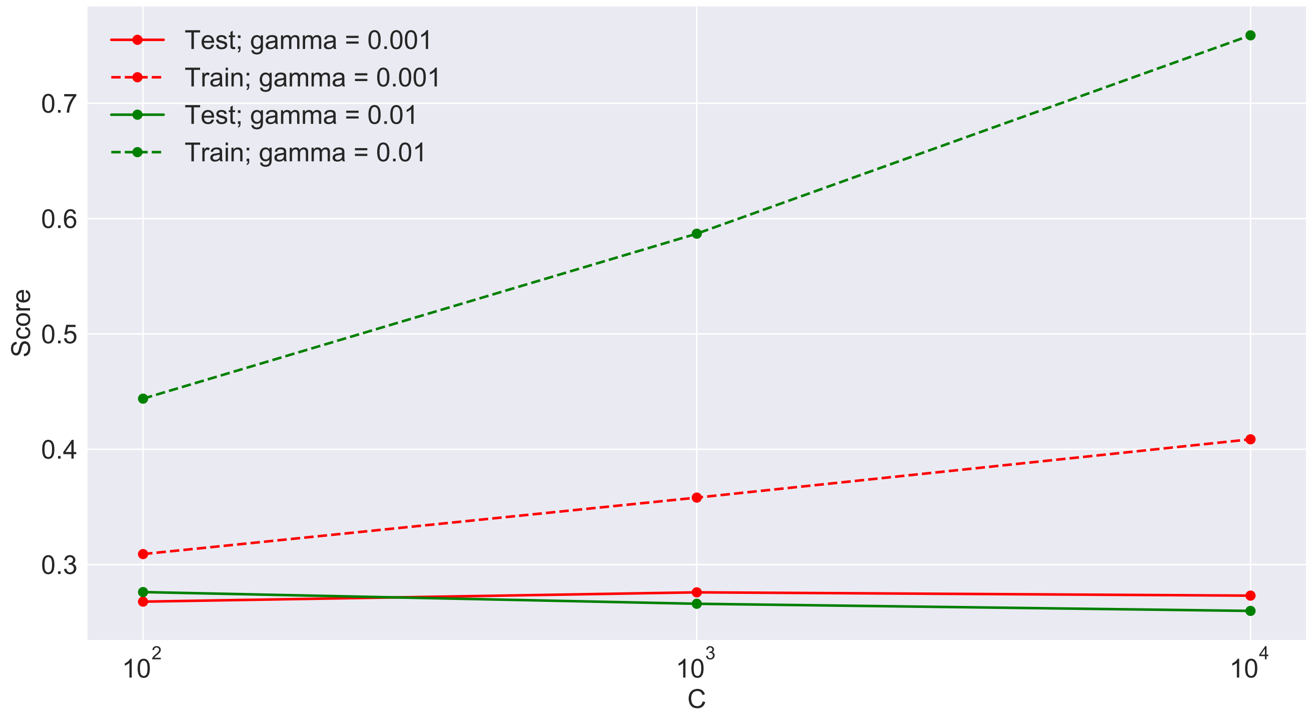


Figure 11: SVM parameter grid search

We can see that a gamma of 0.01 and C of 100 gives the best model. The overfitting is also not too large for gamma of 0.01, which is reassuring. Significant differences between the train and test set scores can imply large overfitting and a poor model.

Tree-based ensemble methods

We can move onto another popular algorithm: Random Forests (RF). Multiple decision trees are grown from the data with replacement. Instead of using all features to chose a node to split, a random subset of features is used instead. This is an advantage as it decreases the variance of the model. A random forest has many (hyper)parameters that we can try to traverse. A similar method is an Extremely Randomized Trees (ERT) algorithm which takes the randomness one step further than a Random Forest. In addition to using a random subset of features, random thresholds are used for the splits as well.

A plot like the one for SVM is much harder for this model because of that reason, as there are too many parameters to show in a single plot. In this particular analysis, three parameters are covered: number of trees, maximum depth of a tree, and minimum number of samples needed to split a tree node. Because of the complexity of this sort of model, interpretation can be somewhat difficult. However, an upside is the ability to calculate feature importance. In other words, a RF and ERT model can calculate which features are valuable and which ones are not in determining the outcome. The results of both the RF (red) and ERT (green) model are shown below in Fig. 12. The standard deviation for each point is shown as the vertical error bars. Both show similar results, though with some notable differences.

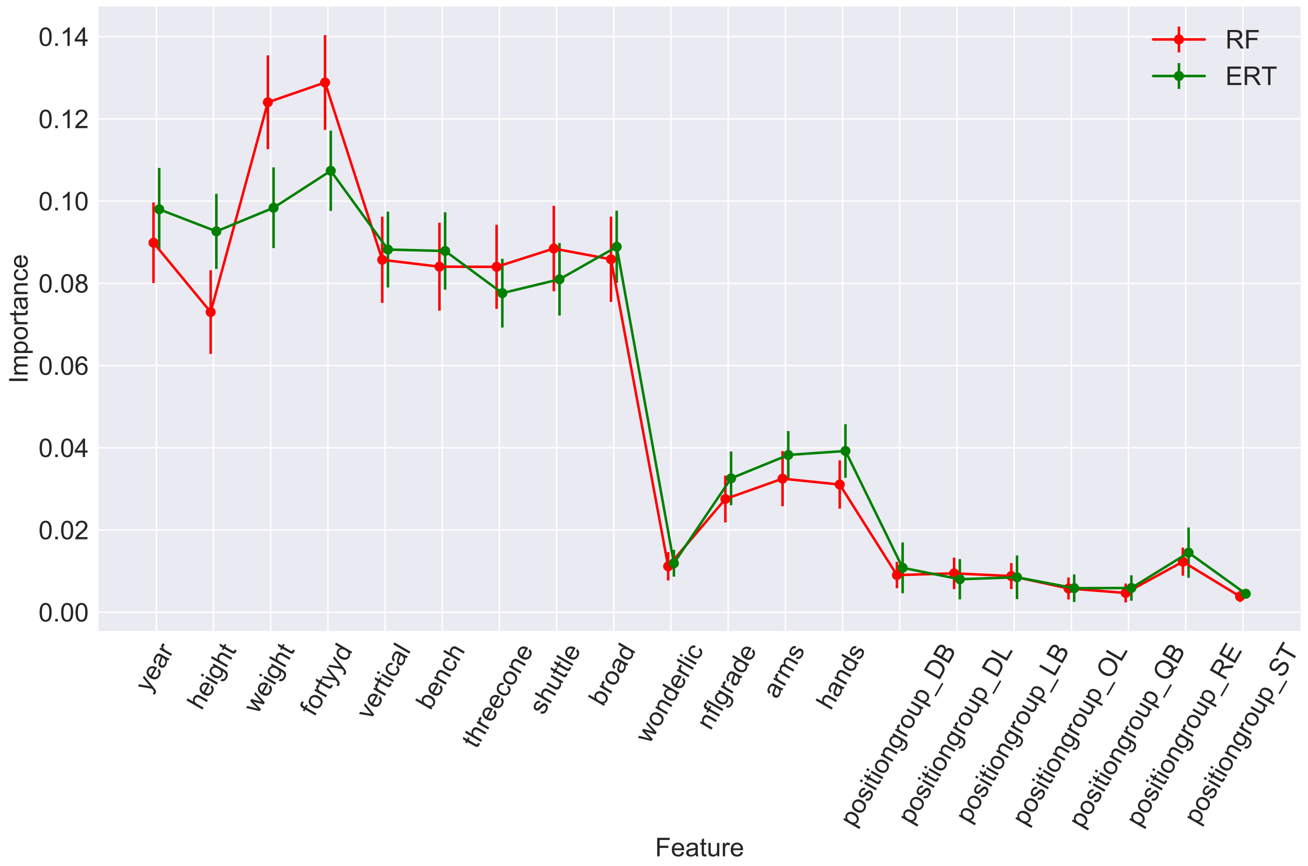


Figure 12: Feature importance from tree-based ensemble methods

The RF pays greater attention to the weight and fortyyd features on the right side of the plot (from year to broad). The ERT meanwhile seems to give near equal weight to all of those features, except fortyyd. Both agree that the fortyyd is an important feature to predict where a player might be drafted. This conclusion agrees with the earlier Logistic Regression coefficients heatmap in Fig. 10. In fact both plots seem to be in agreement on which features likely are important and which ones likely are not.

Stacked ensemble

The last classifier we will test is a majority rules classifier built from the other models we have created. That is, we build a new stacked ensemble method from the other models in this notebook. Each classifier is given a vote for predictions, with a majority rule deciding the outcome. This has advantages of balancing out weaknesses in our models while retaining their strengths. Because our models do not have very different F1 scores, it is worth trying this to see if additional predictive power can be bought by this method. Any potential model instabilities can be ironed out with this type of technique. We can use *scikit-learn*'s `VotingClassifier` to implement this ensemble method.

While not discussed in this document (see the model building notebook, here), a k-Nearest Neighbors model was applied to the data. It seemed to suffer from large overfitting, making it difficult to trust. Therefore, this model is not used in this stacked ensemble. In addition, the Extremely Randomized Trees classifier is also left out. It performed slightly worse than its simpler cousin, the Random Forest, and it follows that only the best version of the two tree-based ensembles should be used here.

Note also that we will use the predicted probabilities from the classifiers and not the prediction itself when voting. Remember that we tested five models and will not use two of them. This leaves three classifiers: Logistic Regression, Support Vector Machine, and Random Forest. This means that using the predictions themselves from the classifiers when voting might result in a tie. However, using the predicted probabilities mitigates this concern and avoids this ambiguity.

We can optimize this model, to a degree, as well by giving some models more voting power than others. That is, we can provide voting weights so that certain classifiers have greater, non-equal say in the final outcome. A quick brute-force grid search over several weight values reveals that the Random Forest model should be weighted more than the other two models.

Results

Now that we have built all of our classifiers, we can finally compare the results of each. Tab. 5 below contains these results. Recall the dummy classifier is the naive model guessing the most frequent class for all inputs, in this case undrafted or -1. This is the baseline mark we want our models to beat. Fortunately, all models do indeed improve upon this score. Using F1 score, the best individual classifier is Random Forest. But the ensemble method is able to improve on all the individual component models and has the best score. Because it performs best, this ensemble classifier is the final model we will use.

Table 5: Classifier cross-validated scores

Classifier	F1 score
Dummy	0.1936
Logistic Regression	0.2416
k-Nearest Neighbors	0.2479
Support Vector Machine	0.2309
Random Forest	0.2907
Extremely Randomized Trees	0.2797
Stacked Ensemble	0.2911

As a visual check of the performance of our final model, we can construct a confusion matrix, in Tab. 6, to see the distribution of the predicted classes vs the actual classes. This makes it easy to see where the model is getting things correct and where it is getting it wrong.

Table 6: Confusion matrix for stacked ensemble classifier

Predictions \ Actuals	-1	1	2	3	4	5	6	7
-1	294	9	24	33	57	52	47	50
1	33	71	29	37	18	6	17	16
2	17	8	12	0	10	15	5	5
3	23	12	13	15	7	8	5	4
4	23	5	11	13	18	8	5	5
5	15	4	11	7	8	11	10	6
6	11	0	7	6	3	4	6	3
7	21	2	4	4	2	5	2	11

It is clear the classifier likes to predict that a player goes undrafted a lot. It tends to get players in the higher rounds more incorrect as well. This makes some amount of sense as there can be little difference between a player drafted in the last round and a player not drafted at all. Given that we are trying to predict eight (unbalanced) classes, the stacked ensemble seems to be performing relatively well.

The full code detailing this in-depth analysis and machine learning can be found on my github. A link to this $JupyterNotebook$ is given here.

Conclusion

In this project we used NFL Combine data to predict which round a player is drafted in the NFL Draft. This is then a multi-label classification problem with eight classes: seven draft rounds plus an undrafted class. We use three data sources: the NFL Savant website for much of the Combine data, the Sports Reference website for Combine data completeness, and the DraftHistory website for Draft results completeness. These data sources are merged together into one cohesive dataset. Missing values are imputed (see Tab. 1). Exploratory data analysis is performed across the dataset to better understand the relationships across the datasets. Particular emphasis was paid to correlations with the target variable, draft round (see, for example, Fig. 5). Multiple classifiers are trained to model the data: Logistic Regression, k-Nearest Neighbors, Support Vector Machine, Random Forest, Extremely Randomized Trees, and a stacked ensemble. In the end a stacked ensemble performs best. It is a majority rule classifier built from the other models, in this case the Logistic Regression, Support Vector Machine, and Random Forest models. Using a F1 score as our metric, the stacked ensemble outperforms the baseline 0.29 to 0.19. Because this problem is an eight-class classification problem, this is a pretty good score.

Further work

While we were able to make a model improving the baseline, there are avenues for improving this project on a whole. Additional data sources can be added to our dataset. A list of some of these sources is below.

- player colleges (rankings, conferences, etc.): Having information about a player's college can help quantify the effect a player's college has in the draft process. Colleges with lots of success can expect to have lots of its players drafted by the NFL. In fact for many colleges, this is a selling point to their own recruiting. The athletic conference a college is in can be an important factor as well.

- player stats in college: The stats a player compiles while at the college level is also certainly a factor. Lots of positive stats for the position can speak well to a player being successful at the professional level. A downside is that there are a variety of stats for the various positions meaning there will be legitimate zero values scattered throughout this feature (or features).

- player honors/awards in college: There are a variety of college awards and honors given at the end of the season. Various organizations give out player of the year awards for positions. There are a few "all-star" type games with the best college players, in addition to conferences naming "all-conference" players. These awards could potentially be added up, in some fashion, to give a feature that is number of honors received for a player.

We could also try to add additional data, as we only have information from1999 to 2015. More data can help our models sort out the relationships in our data. It would also increase the available training and testing data so that both phases of model building have greater data available to them. This can help ensure better quality and more stable classifiers.

In addition, more time can be spent on the machine learning part. Additional algorithms can be tested also. Potential algorithms include Extreme Gradient Boosting, using the *xgboost* package, as well as artificial neural networks using the $Keras$ package and a $Tensorflow$ backend. The existing models can be tested with more comprehensive grid searches, probing larger parameter spaces to find the optimal parameters. It must be noted that this later method needs to be done carefully. If a parameter space is too large, it can lead to an artificially high training score by chance alone resulting in overfitting.