CSCI 184 Final Project

Ryan Chappelle, Michael Nocerino, Esteban Samayoa Mayora, Dhwanil Nikunj Mehta



***Introduction***

This applied machine learning project works with a dataset containing the stats of individual NBA players from the 2022-2023 season. Two of our main goals with this project were as follows:

1) To identify and group together players who provide similar value through agglomerative clustering

2) To build machine learning and deep learning models to classify players into an appropriate salary range based on their individual stats and performance.

Throughout this project, we utilized methods from the course such as null value imputation, one-hot encoding, and feature engineering. We also trained both binary and multiclass classifiers using ML algorithms such as Gaussian Naive Bayes, Decision Trees, Random Forest, and Neural Networks throughout this process. The experimentation process was rigorous, and we became more familiar with the various algorithms and experimented with hyperparameters for each of the models. We analyzed the performance of each model by considering numeral evaluation metrics, specific misclassifications where the model failed, and our own subject knowledge on NBA basketball to gauge how well the classifier was performing. This final report provides background information on NBA basketball and the dataset while also providing an in-depth overview of the methods we utilized to guide our decisions throughout the various stages of the data science pipeline for this project. Finally, the report closes with a summary of our results from the modeling and key findings we have taken away.

Technologies and libraries utilized:

Python 3, Google Colab, Pandas, Sci-kit Learn, Scipy, TensorFlow, Keras

***Background Information***

A maximum NBA contract (will be referenced throughout this report as “Max” or “Max Contract”) is defined as being the maximum a player can receive from the team they play for. The cheapest a max contract can be is 25% of the team’s total salary cap, which for the 21-22 NBA season (the season for our dataset) was $30,351,780 (Adams). However, this number was an estimate, as it can be difficult to predict what this number will be since some teams will have different salary caps. For simplicity we will use $30,000,000 as the baseline for a max contract.

***Related Work***

Many of the models driving front office decision making across the NBA are private and unavailable to the public. In 2023, almost every team has data science professionals and analytics departments working on projects to improve their team, play strategy, and chances of winning a championship. Some of the most acclaimed data professionals in the NBA come from organizations such as the Golden State Warriors, Oklahoma City Thunder, and Boston Celtics. Outside of individuals working for professional sports teams, additional research on mathematical modeling to evaluate player performance has been conducted by academic researchers. One paper called *Estimating NBA players salary share according to their performance on court: A machine learning approach* ([link](https://arxiv.org/abs/2007.14694)) discusses how most models to predict salary use linear regression whereas machine learning has been less utilized thus far in this field. This paper comes from researchers at the University of Crete in Greece and uses data from the 2017-2019 NBA seasons. This paper has a similar goal, but only utilizes random forests to go about solving this problem.

***Dataset***

The dataset for this project contains the individual player stats of professional basketball players in the National Basketball Association (NBA) for the 2022-2023 season. The final dataset uses [Per Game Player Stats](https://www.basketball-reference.com/leagues/NBA_2023_per_game.html), [Advanced Stats](https://www.basketball-reference.com/leagues/NBA_2023_advanced.html), and [Player Contracts](https://www.basketball-reference.com/contracts/players.html) from the 2022-2023 season. All datasets come from basketballreference.com and the individual sets are accessible via the hyperlinks above. The cleaned dataset features 450 entries (representing 450 individual NBA players) and 63 columns. Only three of these features - Player Name, Position, and Team- are categorical while the rest are numerical. Below is a sample of the first five rows of the dataset.



***Data Cleaning***

The file‘22\_23\_Cleaning.ipynb’ contains the Python code used to clean and join the various datasets from basketball reference into the final dataset used for our modeling and analysis. The cleaning process began by joining the individual player per game stats dataset and individual player advanced stats dataset. Since both came from basketball reference.com, the indexing and ordering of the rows was identical, and they could easily be merged. Repeat columns were then dropped.

Next, we had to address the fact that players who had played on multiple teams during the 2022-2023 season appear in repeat entries. For example, players like Russell Westbrook who were traded mid-season would have 3 rows describing their stats: one row for his stats with Lakers, one row for his stats with the Clippers, and one row for his overall season stats. For those players who played on multiple teams, only the row containing their full season stats was kept in the final dataset. The dropping of the repeat rows was achieved with the use of iterrows() in Pandas and a simple Python program as seen below. For traded players, we only kept the row with Team == ‘TOT’ which refers to total.

**

The next step was adding on the salary column for all players in the dataset. This was achieved by creating a dictionary mapping player name to player salary from the salary dataset and then looping through the stats dataset and adding on the appropriate salary for each player. There were 20 missing salaries after this was performed so the rest were filled in manually using data entry in excel. Before exporting the final dataset to a .CSV to be used for our clustering and model building tasks, we restricted the dataset to include only players who had played over 15 games (so that players who had been injured or benched the majority of the 2022-2023 season were left out). This left us with a dataset containing the individual stats of 450 NBA players.

***Handling Null Values and Imputation Techniques***

Next, we had to decide how to best handle the presence of a few null values within our dataset. The imputation and handling of null values can be found in the ‘featureEngineering.ipynb’ file. There were only 9 null values present in the dataset having performed all the cleaning tasks described in the paragraph above. 3P% had 7 rows with an ‘NaN’ value while ‘FT%’ had 2 rows with an ‘NaN’ value.

Upon investigating the players with these null columns, it was discovered that the rows with a null value for three point percentage (3P%) were all centers who play inside the paint and hadn’t attempted a single three pointer during the entire season. Because we didn’t want to unfairly punish these players and give them a three point percentage of 0, the null value was imputed to 0.15. Because we can assume these players aren’t good three point shooters due to them never attempting a three point shot, we imputed with 0.15 – a very poor but non-zero ‘3P%’.

The two players who hadn’t attempted a free throw were Ryan Arcidiacono and PJ Dozier. Because these were two guards who had displayed decent shooting in their previous seasons, the null value was imputed with the league average free throw percentage (complete[‘ FT%’].mean() ).

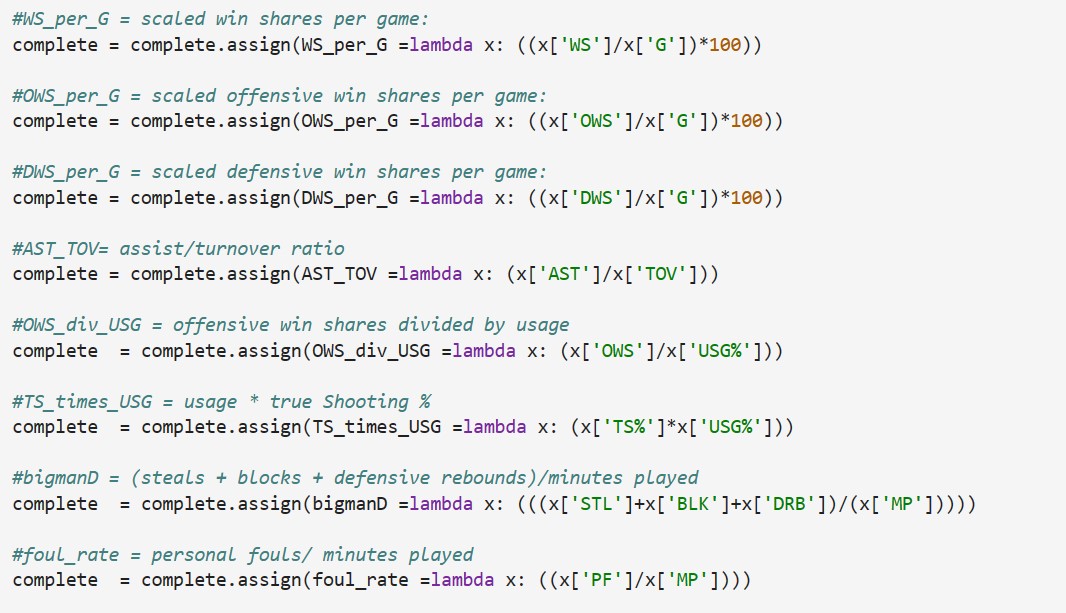
***Feature Dictionary***

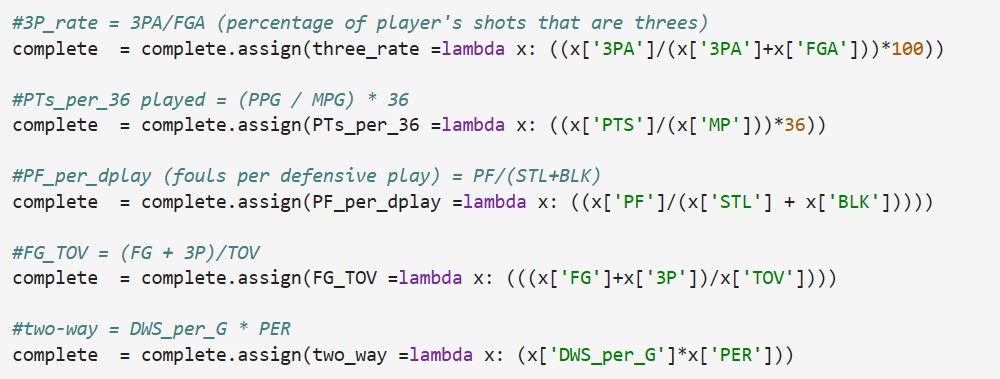
Below is a feature dictionary containing all 50 original features from basketballreference.com as well as a description of the abbreviation.

| Feature Name | Description |
| --- | --- |
| Player | Player First and Last Name |
| Pos | Position |
| Age | Age |
| Tm | Team |
| G | Games Played |
| GS | Games Started |
| MP | Minutes Per Game |
| FG | Field Goals Made Per Game |
| FGA | Field Goal Attempts Per Game |
| FG% | Field Goal Percentage |
| 3P | Three Pointers Made Per Game |
| 3PA | Three Point Attempts Per Game |
| 3P% | Three Point Percentage |
| 2P | Two Pointers Made Per Game |
| 2PA | Two Point Attempts Per Game |
| 2P% | Two Point Percentage |
| eFG% | Effective Field Goal Percentage |
| FT | Free Throws Per Game |
| FTA | Free Throw Attempts Per Game |
| FT% | Free Throw Percentage |
| ORB | Offensive Rebounds Per Game |
| DRB | Defensive Rebounds Per Game |
| TRB | Total Rebounds Per Game |
| AST | Assists Per Game |
| STL | Steals Per Game |
| BLK | Blocks Per Game |
| TOV | Turnovers Per Game |
| PF | Personal Fouls Per Game |
| PTS | Points Per Game |
| PER | Player Efficiency Rating |
| TS% | True Shooting Percentage |
| 3PAr | Three Point Attempt Rate |
| FTr | Free Throw Rate |
| ORB% | Offensive Rebound Percentage |
| DRB% | Defensive Rebound Percentage |
| TRB% | Total Rebound Percentage |
| AST% | Assist Percentage |
| STL% | Steal Percentage |
| BLK% | Block Percentage |
| TOV% | Turnover Percentage |
| USG% | Usage Rate |
| OWS | Offensive Win Shares |
| DWS | Defensive Win Shares |
| WS | Win Shares |
| WS/48 | Win Shares Per 48 Minutes |
| OBPM | Offensive Box Plus-Minus |
| DBPM | Defensive Box Plus-Minus |
| BPM | Box Plus-Minus |
| VORP | Value Over Replacement Player |
| Salary | Salary (in Millions) |

***Feature Engineering***

Feature engineering was used to create thirteen new features that could potentially provide new insight into player performance and evaluation. The snippet of code below contains a series of lambda functions in Python that were used to create the engineered features. An in depth feature dictionary of the engineered features also follows the code excerpts.

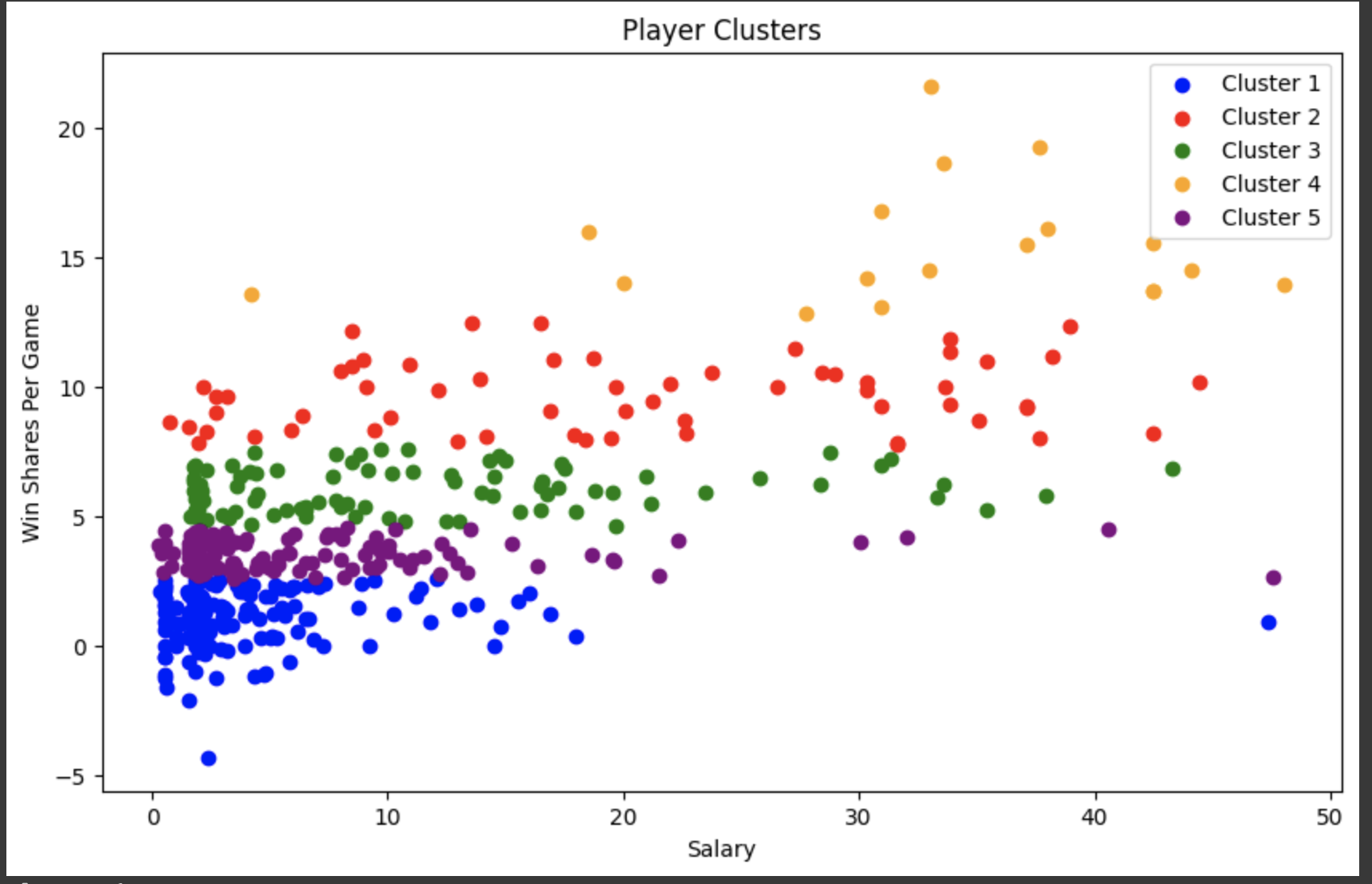




| Feature Name | Description | Derivation |
| --- | --- | --- |
| WS\_per\_G | Win Shares Per Game | WS/G |
| OWS\_per\_G | Offensive Win Shares Per Game | OWS/G |
| DWS\_per\_G | Defensive Win Shares Per Game | DWS/G |
| AST\_TOV | Assist-Turnover Ratio | AST/TOV |
| OWS\_div\_USG | Offensive WS divided by Usage | OWS/USG% |
| TS\_times\_USG | True Shooting % Times Usage | TS% \* USG% |
| bigmanD | Bigman Defensive Metric | (STL + BLK +DRB)/MP |
| foul\_rate | Personal Foul Rate | PF/MP |
| three\_rate | Three Point Rate | (3PA/(3PA + FGA))\* 100 |
| PTs\_per\_36 | Points Per 36 Minutes | (PTS/MP)\* 36 |
| PF\_per\_dplay | Fouls Per Defensive Play | PF/(STL+BLK) |
| FG\_TOV | Field Goals Per Turnover | (FG+3P)/TOV |
| two\_way | Two Way Player Metric | DWS\_per\_G \* PER |

***Clustering: Visualizing Player Value and Identifying Similar Players***

We used agglomerative clustering to find out what players have higher win\_shares\_per\_game along with the salary they earn. This clustering helps us to understand if a player is being overvalued or undervalued. The clusters which were built from this algorithm gave a clear picture of what players were being paid fairly or not. For example, players like Terrence Ross are being paid a high salary but he has a very low win\_shares\_per\_game. On the other hand, Tyrese Haliburton has a low salary but has a high win\_shares\_per\_game. For key players like Lebron James, Nicola Jokic and Luka Doncic the algorithm assigns them in the right clusters where they have a high salary as well as a decent to high win\_shares\_per\_game. The clustering algorithm also gives a great idea on how players are performing based on their win\_shares\_per\_game and simultaneous features for that player can be seen on the dataset to see where they can improve their performance based on those features of the data. We used a scatterplot to depict the clusters and the figure displays the clusters based on win\_shares\_per\_game and the salaries of the players. We then stored the clustering data to a [csv file](https://drive.google.com/file/d/1C-kV6t5fFiwF0zxhqni8f74m0vNLMDCa/view?usp=drive_link) to display all the players in the data.



In addition to clustering with Win Shares Per Game and Salary, we also performed hierarchical clustering using the important features 'FG%', '3P%', 'TS%', 'TRB', 'AST', 'TOV', 'two\_way', 'DWS\_per\_G', 'OWS\_per\_G', 'USG%', 'OWS\_div\_USG' to see how players were clustered based on these statistics. The dataset was limited to star players averaging over 20 points per game, and this star clustering was performed with ward linkage (one of the options in Sci-kit Learn), though we also experimented with single, complete, and average linkage in the ‘clustering.ipynb’ file.

| 0 – Bojan Bogdanović, Jordan Clarkson, Jerami Grant, Jalen Green, Keldon Johnson, Anfernee Simons |
| --- |
| 1 – Bradley Beal, Kyle Kuzma, Tyrese Maxey, Kelly Oubre Jr., Jordan Poole, Terry Rozier, Klay Thompson |
| 2 – Giannis Antetokounmpo, Anthony Davis, Joel Embiid, Nikola Jokić |
| 3 – Devin Booker, Stephen Curry, DeMar DeRozan, Anthony Edwards, Darius Garland, Paul George, Julius Randle |
| 4 – Luka Dončić, Jayson Tatum |
| 5 – Bam Adebayo, Jaylen Brown, James Harden, LeBron James, Kawhi Leonard, Ja Morant, Kristaps Porziņģis, Zion Williamson |
| 6 – LaMelo Ball, Mikal Bridges, Jalen Brunson, De'Aaron Fox, Tyrese Haliburton, Tyler Herro, Lauri Markkanen, CJ McCollum, Dejounte Murray |
| 7 – Desmond Bane, Brandon Ingram, Kyrie Irving, Zach LaVine, Pascal Siakam, Karl-Anthony Towns |
| 8 – Jimmy Butler, Kevin Durant, Shai Gilgeous-Alexander, Donovan Mitchell |
| 9 – Damian Lillard, Trae Young |

We were pleased at the results of the clustering and think it can provide valuable insight for NBA front office decision makers who are looking to acquire players who can provide similar value to some of the league’s biggest stars. It is fascinating to see the player similarities that we can visualize through clustering, and the clusters are highly effective in grouping similar players.

***Player Salary Classification with Machine Learning:***

*Gaussian Naives Bayes*

Introduction:

A Naïve Bayes Classifier is a machine learning model that treats each feature as independent from each other, thus the term naïve. For our case the goal was straightforward for this model: after being trained on the dataset, classify each test example into a certain salary range. The initial implementation is a binary classifier that decides if a player is deserving of a “max” contract or not.

Binary Classifier:

The first step after loading in the dataset was to add the “Class” column to it, which held values of “Max” or “Not Max” depending on their salary for the season. Then we dropped this new column from the dataset and stored it into a variable y which would be our target. After the dataset was ready, I needed to do a little work to optimize the train\_test\_split function. In some cases, I was receiving output where the test accuracy was higher than the train accuracy. The accuracies followed the same trends, but an optimal model would have the highest possible testing accuracy while being smaller than the training accuracy. I ran the model on different random\_state variables from 0-100, and found that the best value for it was 60. After splitting into train and test sets, I dropped any non-numerical values such as player name, team name, and position and stored them for later use. After fitting the model to the training data using a Gaussian Naïve Bayes (which is appropriate for numerical data), the big step of running the model on test data as well as outputting the results was next. This could all be done thanks to sklearn.metrics, using f1 score as the main metric, but also taking into account precision and recall. For the binary classifier I calculated these values first as the weighted average between both classes, and then also individually for each class. Below are the results of the binary classifier.

Fig1. Fig2.

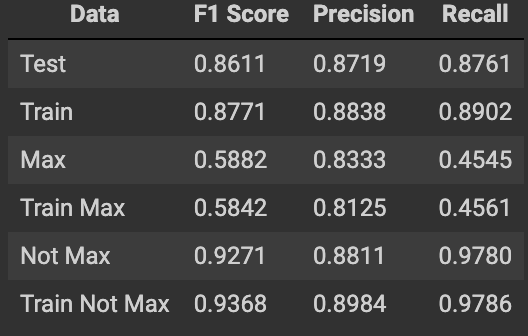
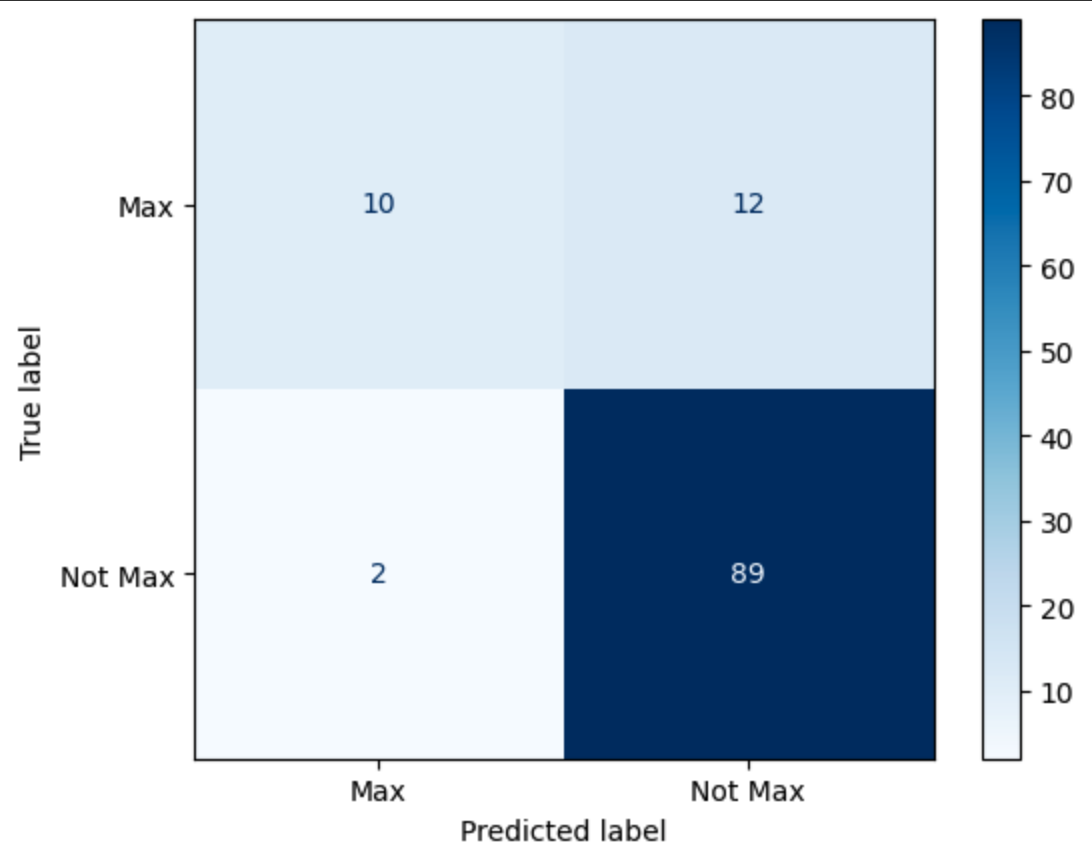


Fig3.

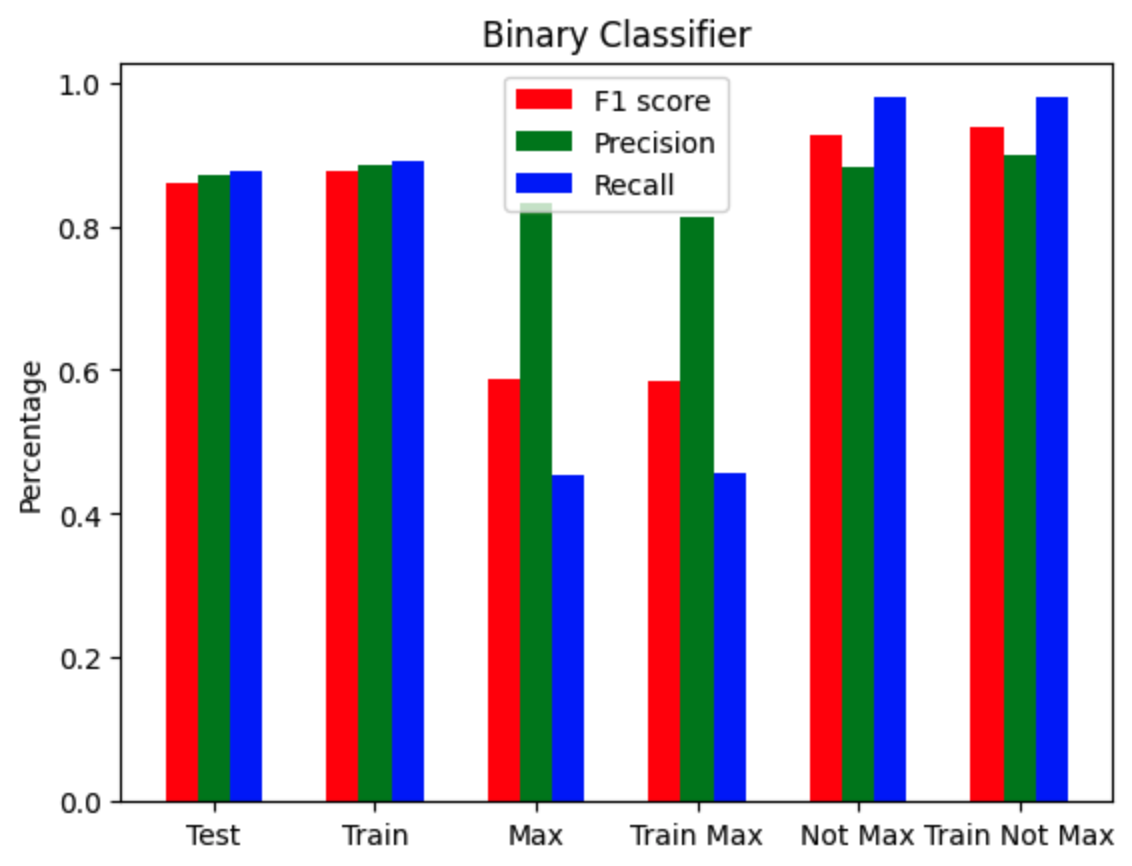


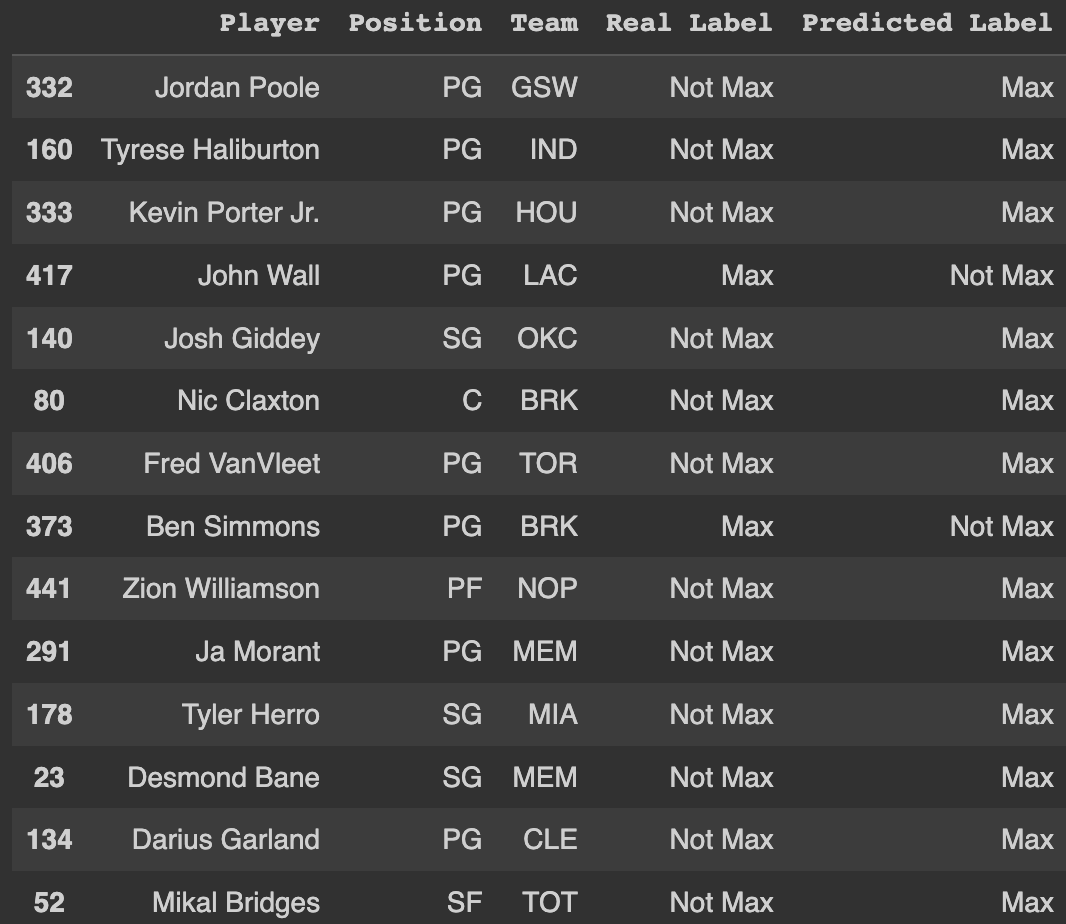
Fig1: Confusion matrix demonstrating the breakdown of how the model predicted each observation compared to its true label.

Fig2: Table representing the accuracy of the model based on several datasets:

* Max: The accuracy of the model when predicting a test set observation whose true label is “Max”.
* Train Max: The accuracy of the model when predicting a train set observation whose true label is “Max”.
* Not Max: The accuracy of the model when predicting a test set observation whose true label is “Not Max”.
* Train Not Max: The accuracy of the model when predicting a train set observation whose true label is “Not Max”.
* Test: Both the Max and Not Max rows are divided by the number of observations they see, thus weighting them equally. Then the halfway point of these two values is chosen as the weighted average. This is a strong metric as it gives an accurate representation of our model’s ability to predict a data point regardless of its true value.
* Train: Same as Test except for training dataset.

Fig3: Graph corresponding to the table in Fig2. We can see that the testing and training performances are very similar, with training accuracies slightly outperforming testing. These accuracies are great overall.

Fig4: Some of the mislabeled players from the testing set.



Multiclass Classifier:

After the binary classifier, I wanted to see if adding another class to the “Class” column would affect performance. The multiclass classifier had a nearly identical implementation. The main difference was that the “Class” column of the dataset needed to be altered to distinguish three separate classes of salary ranges. I chose these to be from $0 to $9mil, $9mil to $30mil, and the highest is just the same “Max” class from the binary classifier. Everything else was implemented the exact same, with minor differences to accommodate the multiclass output instead of binary. The random\_state for train test split was found to be best when equal to 5.

Fig5. Fig6.

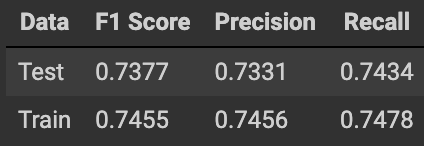
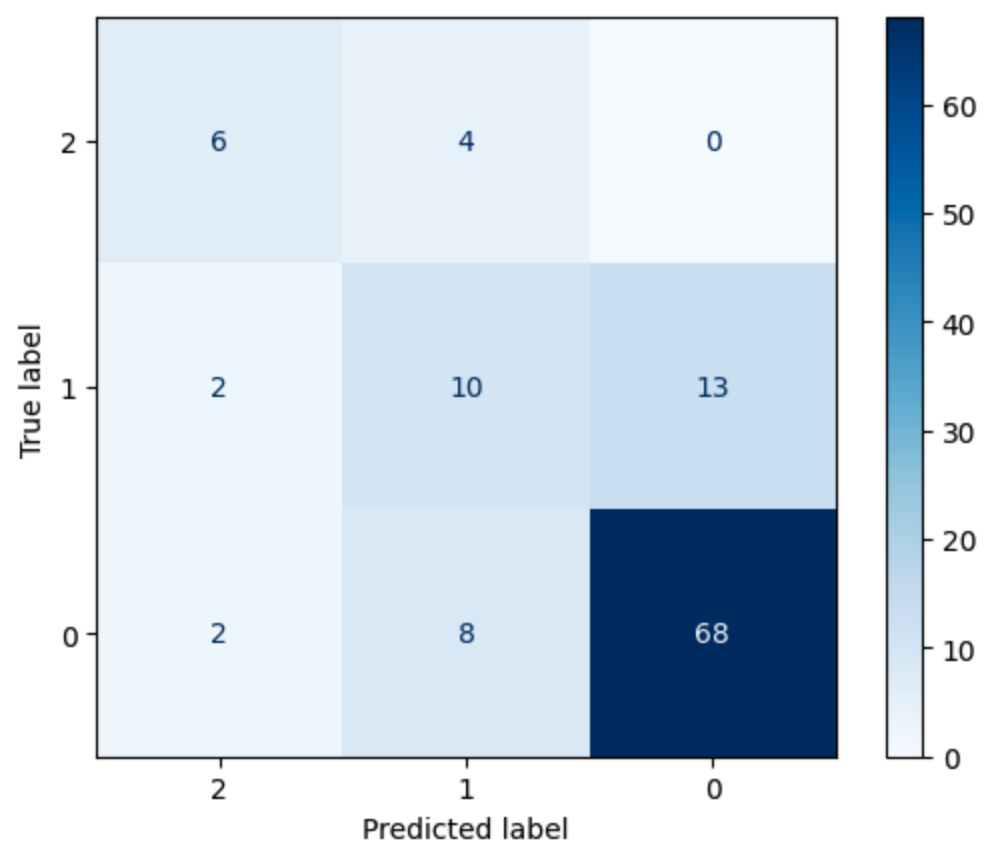


Fig7.

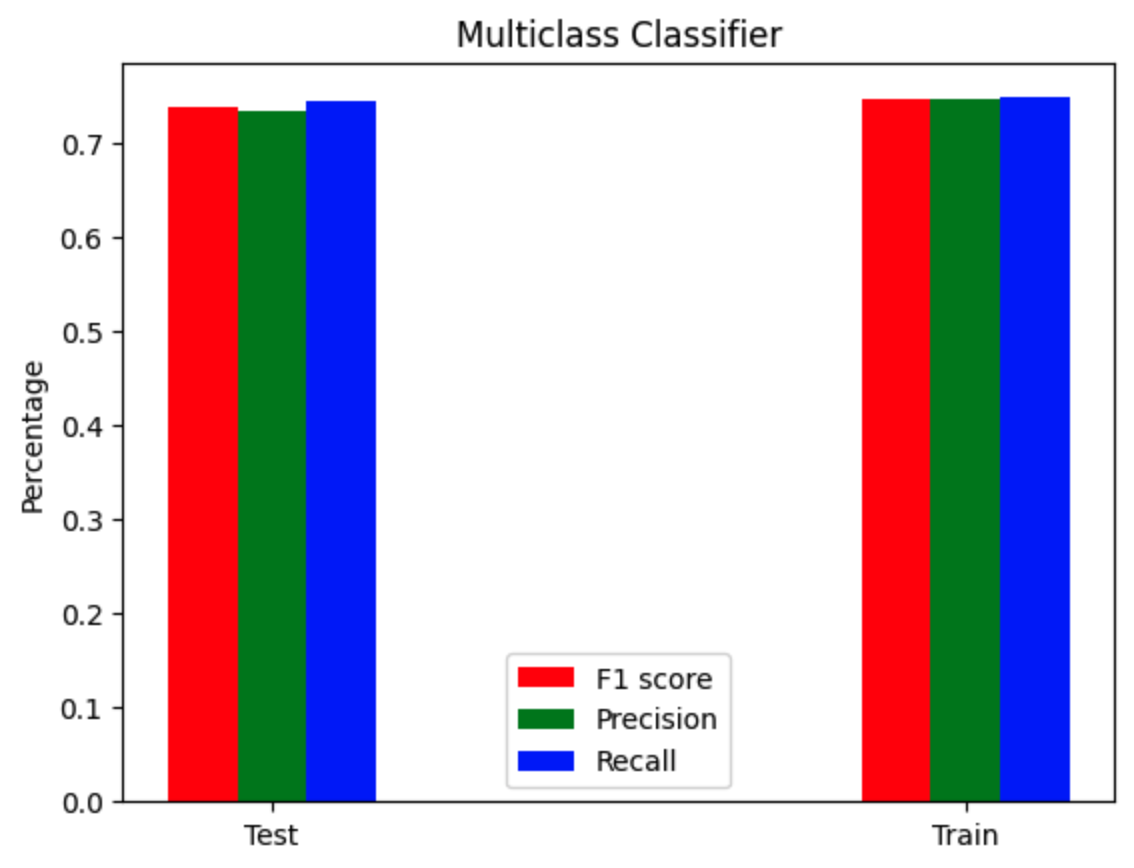
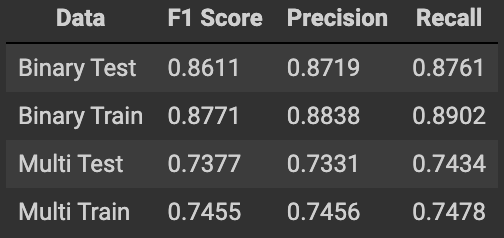


Fig5: Confusion matrix for three classes.

Fig6: Table holding both testing and training data

Fig7: Graph corresponding to Fig6.

Fig8: Table that compares the binary classifier to the multiclass classifier.



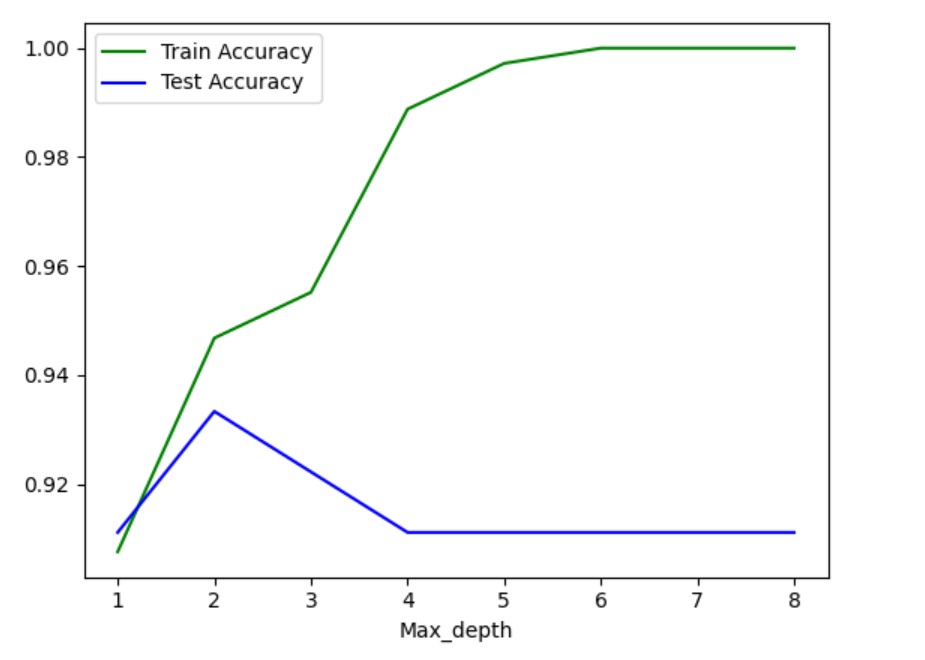
Analysis:

For the binary classifier, we can see from Fig2 that predicting an observation that belongs to class “Max” is far more difficult than class “Not Max”. The overall f1 score though is our most important metric, and this comes in at 86% for test data which is very good. This means that there is roughly an 86% chance that any new observation will be correctly classified. Moving onto the multiclass classifier, it has a much lower overall f1 score for test data. Still, 73% is pretty solid, though maybe not good enough to be reliable. The binary classifier can definitely be used by NBA offices to help determine if a player is deserving of a max contract or not.

*Decision Tree*

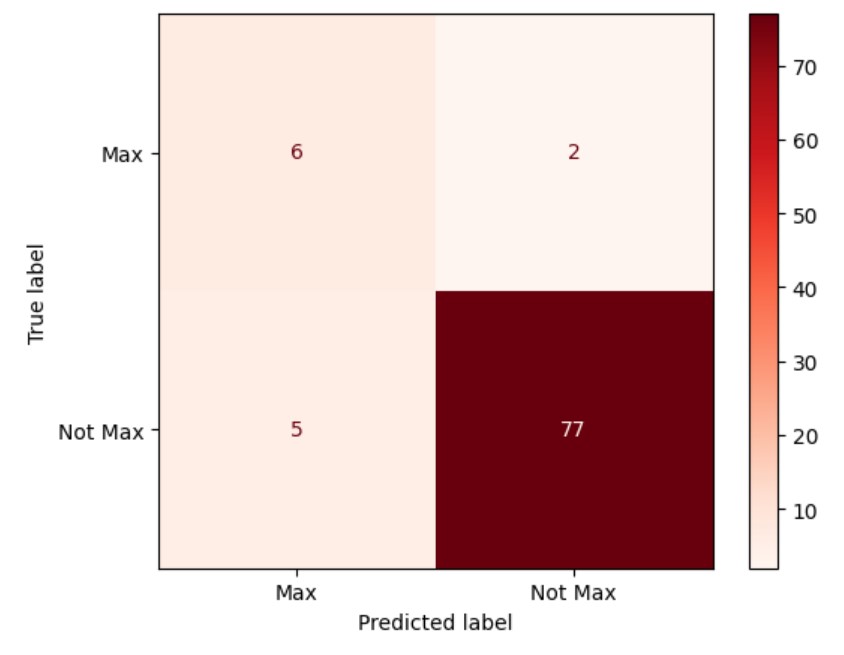
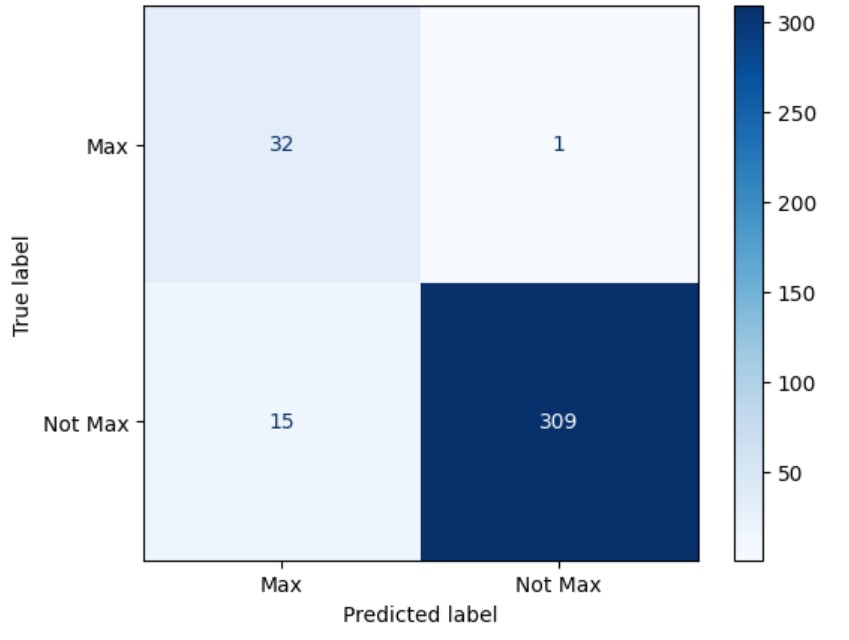
We then experimented with the decision tree algorithm for binary classification and used the same class brackets used above with ‘Not Max’ being a player making less than 30 million and ‘Max’ any player making over 30 million. The decision tree code can be found in the ‘dt.ipynb’ file, and a 80-20 train test split of the original dataset was utilized for training the model.

In order to tune the hyperparameter max\_depth (which is a method for pre-pruning the decision tree), we graphed the train & test accuracy vs. max\_depth which can be seen in the figure below. The graph suggests that a max depth of 2 or 3 should be selected to avoid heavily overfitting to the training data. We went ahead and decided to select max\_depth = 3 because we thought having a max\_depth of 2 would be too simple of a tree.



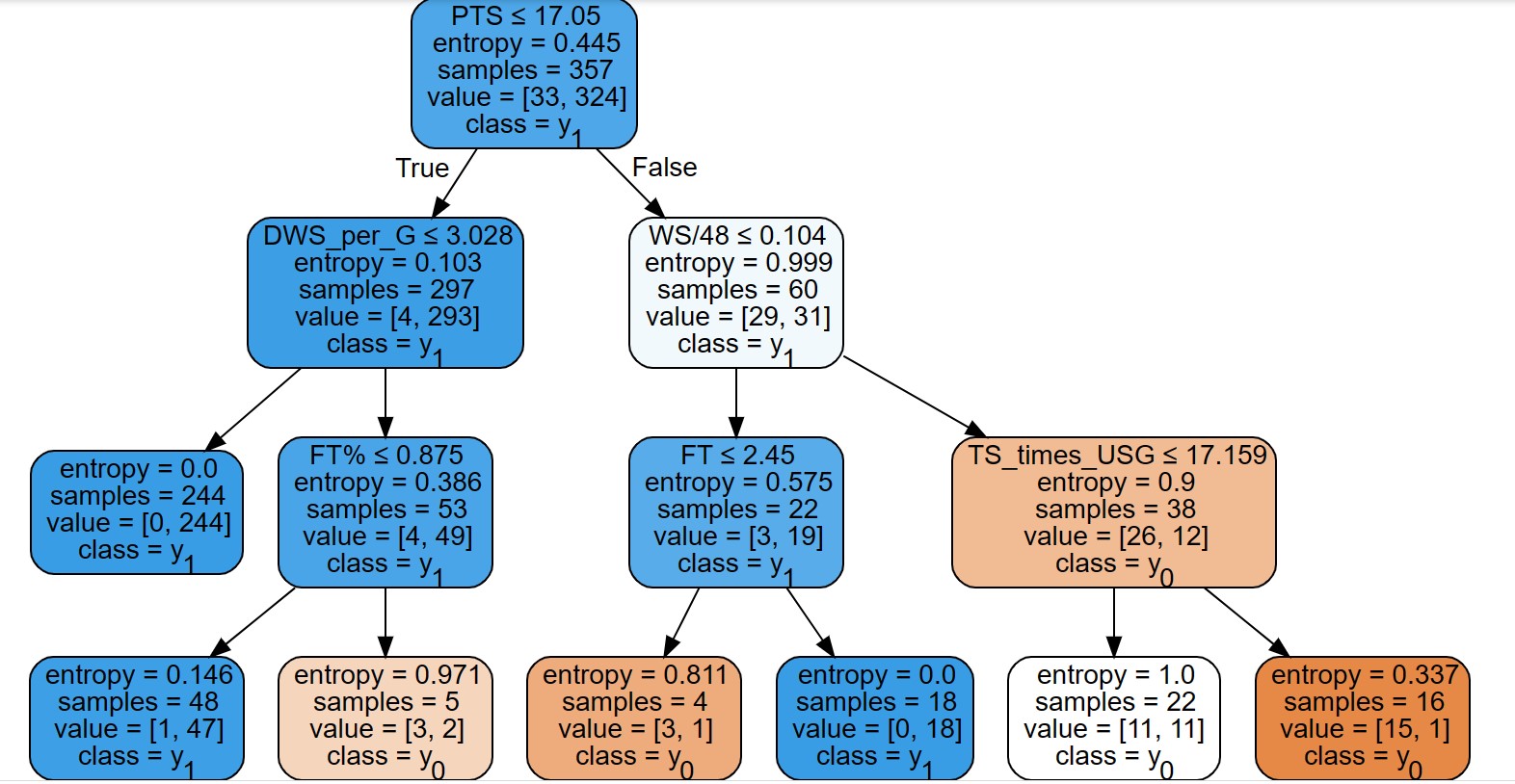
The features that appeared in our final tree were PTS, DWS\_per\_G, WS/48, FT%, FT, and TS\_times\_USG. The blue confusion matrix below shows the performance of the decision tree classifier on the training data whereas the red matrix shows how the model performs on unseen test data. The entire tree with the various thresholds for each feature can be seen as well on the page that follows. One issue we encountered with the decision tree algorithm and this dataset was producing a complex, insightful model that wasn’t overfit. As you can see from the training confusion matrix, the model learned the training data very well in terms of identifying which Not Max players deserve a max so it is successful in identifying underpaid players. However, it did not learn to classify many existing Max players as Not Max so it may struggle to identify unseen players that are overpaid.

Misclassifications aren’t necessarily a bad thing within this project in which we are seeking to build a model that accurately represents which players are worthy of a max contract. Because the dataset we are training on has plenty of outliers – including both overpaid and underpaid players, the final model would ideally make some misclassifications which helps us identify which contracts are poor – player is overpaid or underpaid.

**

Below, are the various evaluation metrics for the decision tree binary classifier:

|  | Accuracy | Precision | Recall | F1-Score |
| --- | --- | --- | --- | --- |
| Train | 0.955 | 0.681 | 0.969 | 0.799 |
| Test | 0.922 | 0.545 | 0.75 | 0.631 |

**

**

Above is a look at the misclassifications the decision tree made on the test set. Applying subject knowledge on these NBA players, most of these ‘mistakes’ are highly reasonable predictions in that many of the players in this misclassifications list are either overpaid or underpaid. Players like Ja Morant, Mikal Bridges, and Jaren Jackson Jr. are widely considered to be players deserving of a max when they receive their next contract. On the other hand, Russell Westbrook is widely considered to be overpaid at this point in his career.

**

We also took a look at the misclassifications the model makes when it is run on the training data as this gives us some insight into how the model learned the training data. Although there are a few poor mistakes such as Grayson Allen and Trey Murphy III, the other misclassifications are valid corrections for underpaid/overpaid players. As we can see from the training data, the model is much more likely to suggest a player is underpaid (has not max but deserves max) than overpaid (has max but deserves not max).

Because the decision trees built were quickly overfitting to the training set even with a small max depth, we weren’t confident that the model would perform well when attempting multi-class classification. We decided to instead try to focus our attention on creating a multi-class classifier using neural networks as described below.

*Deep Learning*

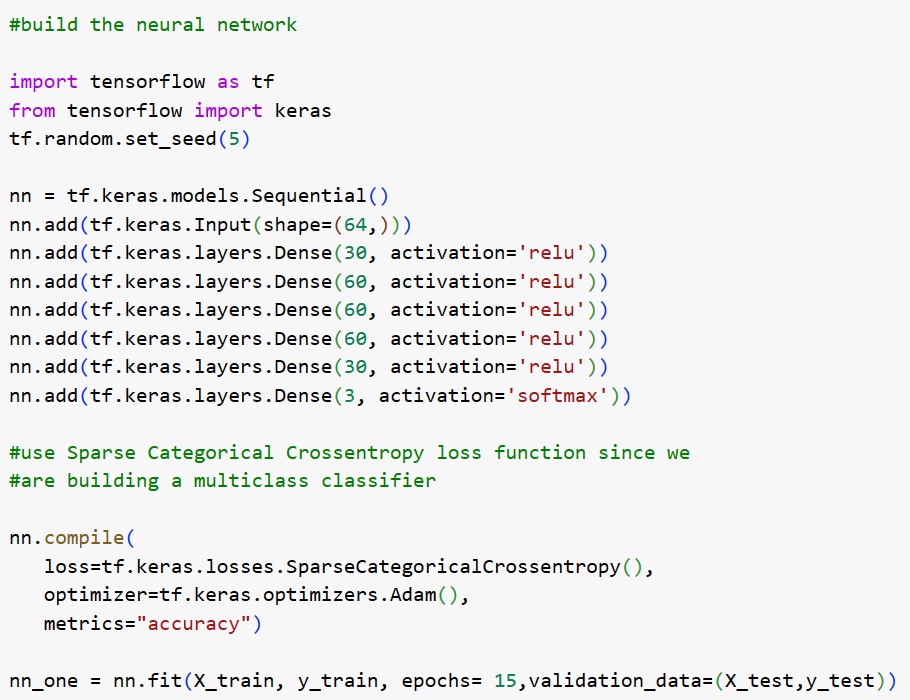
We were able to train a neural network that was highly effective for multiclass classification. The classes used were as follows:

| Low Tier | Salary < 9 million |
| --- | --- |
| Middle Tier | Salary ≥ 9 million & < 30 million |
| Star | Salary ≥ 30 million |

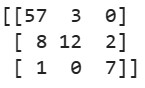
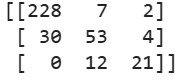
The Neural Network built uses Sparse Categorical Cross Entropy as the loss function and softmax activation for the output layer since we had a multi-class classification problem with 3 classes. We also used the Adam optimization in Tensorflow. Below, we describe the structure of the network and show the code to build it using Tensorflow and Keras.

Neural Network Structure

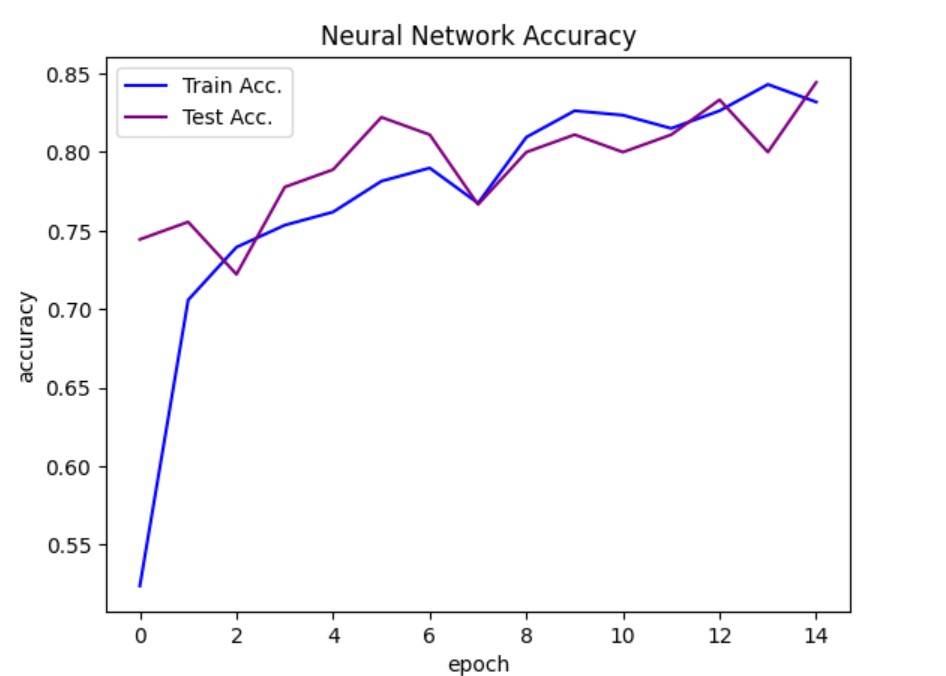
* **Input Layer:** 64 neuron (for all 64 features)
* **Hidden Layer 1:** 30 neurons, ReLu activation
* **Hidden Layer 2:** 60 neurons, ReLu activation
* **Hidden Layer 3:** 60 neurons, ReLu activation
* **Hidden Layer 4:** 60 neurons, ReLu activation
* **Hidden Layer 5:** 30 neurons, ReLu activation
* **Output Layer:** 3 neurons, Softmax activation



The Neural Network was trained for 15 epochs, and resulted in a training accuracy of 0.846 and a testing accuracy of 0.844. The respective confusion matrices and performance scores for the neural network are below (train on the left, test on the right). The graph that follows shows how both training and testing accuracy improve with each epoch of training the network, showing that the neural network did not overfit. Overall, we were very pleased with the results of the neural network for multi-class classification and found that the classifier performed around 85% for accuracy, precision, recall, and f1-score making it extremely well rounded in its performance.

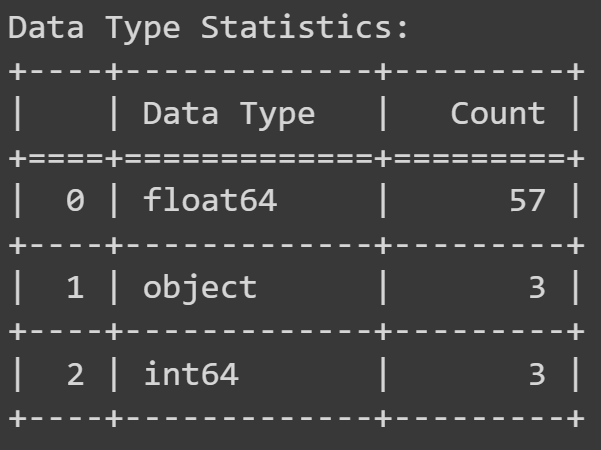


|  | Accuracy | Precision (Weighted) | Recall (Weighted) | F1- Score (Weighted) |
| --- | --- | --- | --- | --- |
| Train | 0.846 | 0.866 | 0.846 | 0.846 |
| Test | 0.844 | 0.875 | 0.844 | 0.854 |



*Random Forest*

*Data Preparation*

**

There are 3 categorical variables in the dataset:

1. Player = player's name (dropped)
2. Pos = player's position (encoded)
3. Tm = player's team (dropped)

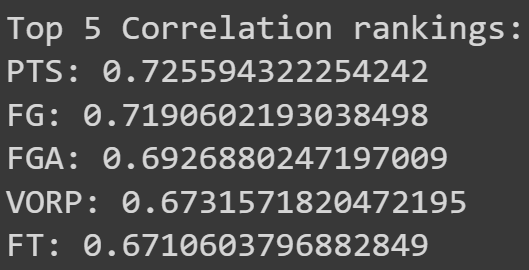
Considering there are 5 positions in basketball, we can use one-hot encoding for that categorical variable. There are 30 teams in the NBA, however studying the value of a player has little relation to the team they play on so that column can be ignored for the study of a player's value. Finally, a player's name is completely irrelevant.

Finally, the last bit of data prep is scaling the target variable, which is salary. For Random Forest Classifier, you need categorical or discrete values for the target variable. Therefore, we will be binning salaries into 8 bins in order to be able to classify the data correctly.

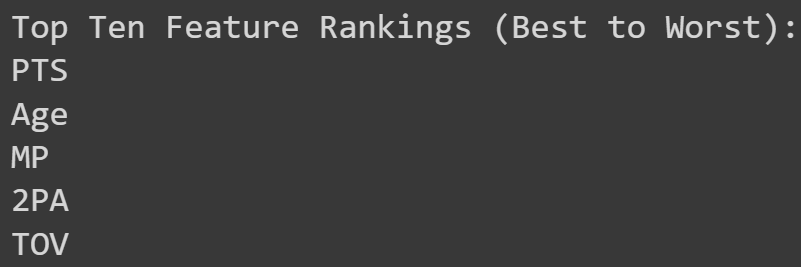
*Feature Selection*

For a random forest, we tried to figure out two different ways of selecting features that would be included in the model, and after that, test each of those feature sets with a different number of trees. Through this process we were able to maximize the accuracy of the models and stay away from overfitting the model within the data.

First, we tried ranking features on basic correlation. These were the top 5 rankings with their respective coefficients:

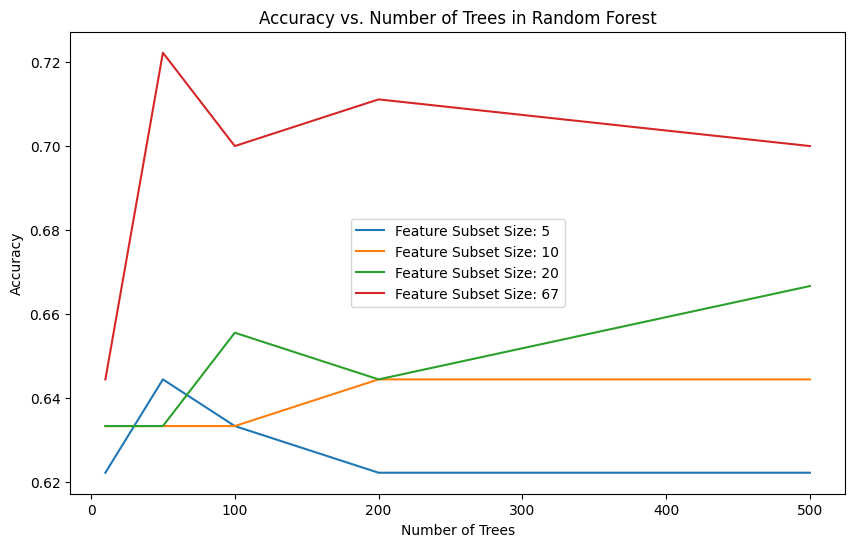


Then, we ranked the features using something called GradientBoosingRegressor



*Basic Correlation - Random Forest Experimentation*

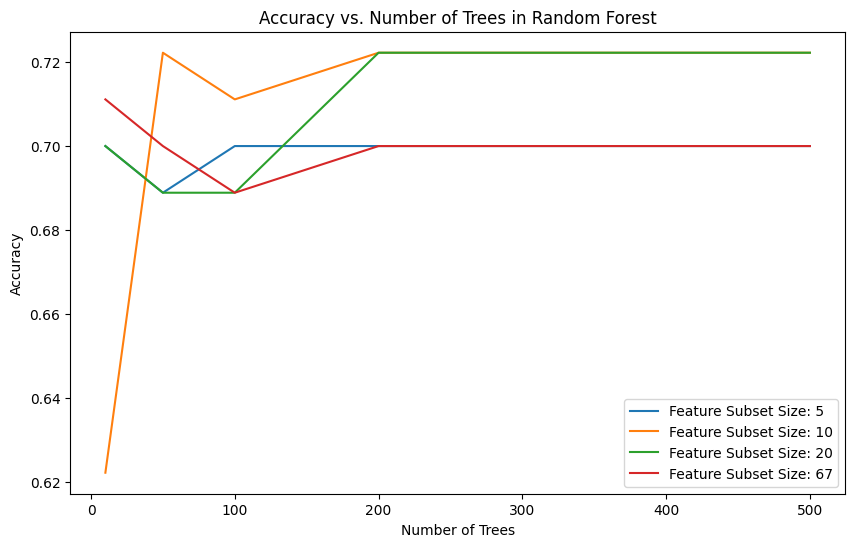
After using the features obtained from basic correlation, and graphing it including number of trees and number of features, these were the results:



It is clear that when taking the features from correlation, the more features included in the random forest model, the more accurate the model will be. The sweet spot is 67 features with 50 trees in the algorithm.

*Gradient Boosting Regressor - Random Forest Experimentation*

After using the features obtained from gradient boosting regressor, and graphing it including number of trees and number of features, these were the results:



A similar accuracy point was reached but except this time it was through a feature subset of size 10. Further experimentation would be required to conclude whether or not these results are good enough for classification, because it does not make enough logical sense given the shape of these lines.

*Random Forest Conclusion*

As a conclusion, for the experimentation that was done, it would be better to go with the first random forest approach, where the features selected for the model come from simple correlation, and then you use all 67 features. In order to fully commit to the Gradient Boosted Regressor, further experimentation would have to be completed.

***Final Results and Interpretations***

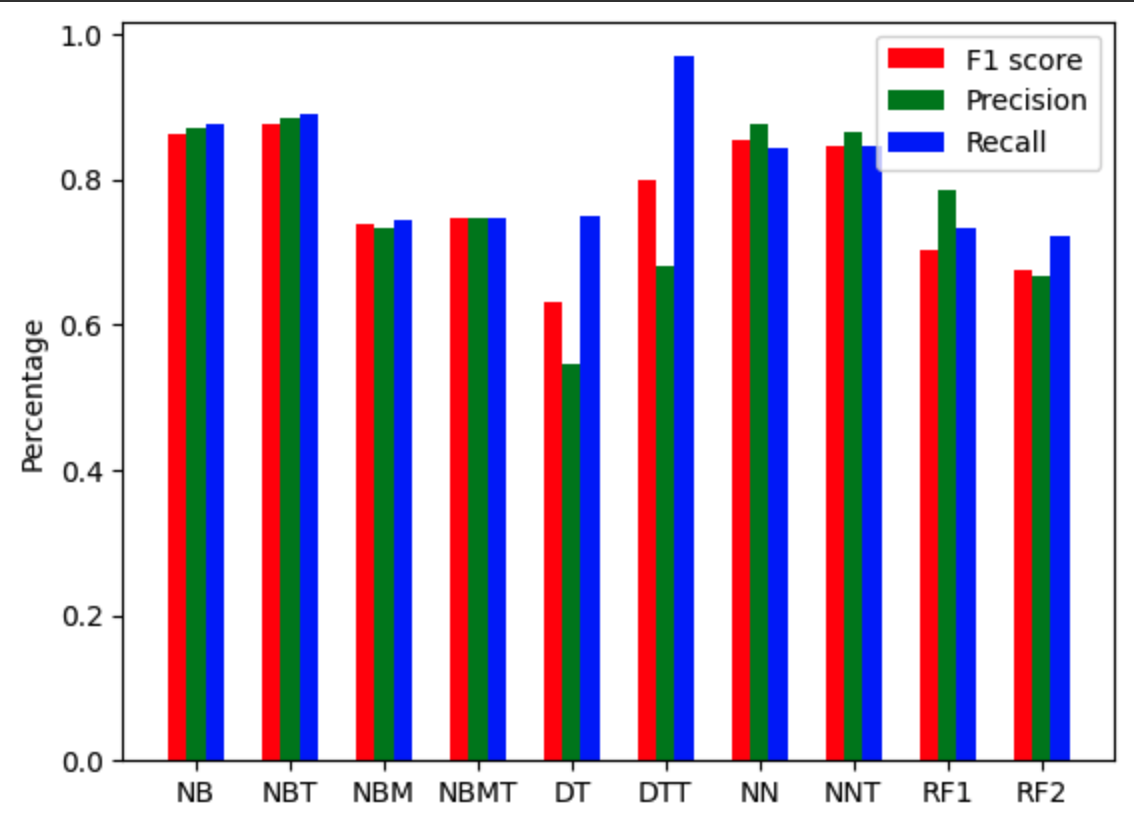
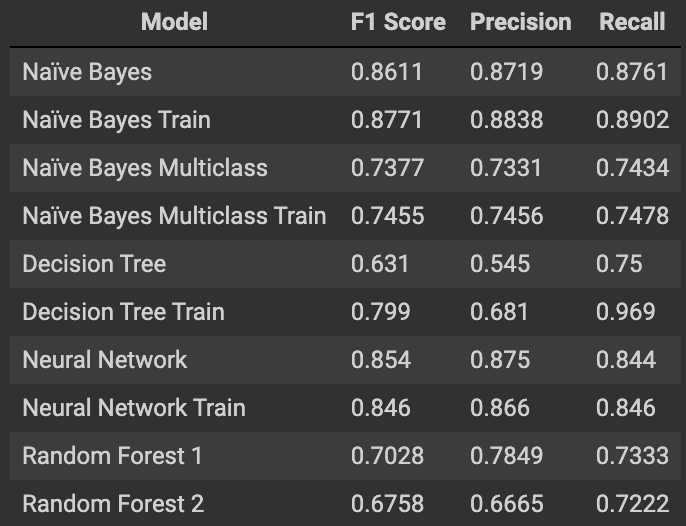
Throughout this project we have used clustering, as well as four separate classification models to fulfill our goals.

1) To identify and group together players who provide similar value through agglomerative clustering

2) To build machine learning and deep learning models to classify players into an appropriate salary range based on their individual stats and performance.

Our first goal was fulfilled by running our dataset through agglomerative clustering with two features being used: salary and win\_shares\_per\_game. This worked very well, and successfully clustered players into five different categories. Additionally, we ran agglomerative clustering on a subset of only NBA superstars to see who would be grouped together. Many more features were included, and this worked extremely well. This first goal was met thoroughly and through different angles.

The second goal also yielded exciting results. Because four different models were used in classifying players, we are able to compare the performance of each of these to see which might be the most desirable. The following is a table holding all of the accuracy scores of these models, followed by a bar chart corresponding to the table.



NB: Naïve Bayes

NBT: Naïve Bayes Train

NBM: Naïve Bayes Multiclass

NBMT: Naïve Bayes Multiclass Train

DT: Decision Tree

DTT: Decision Tree Train

NN: Neural Network

NNT: Neural Network Train

RF1: Random Forest with subset size = 10 and number of trees = 500

RF2: Random Forest with subset size = 20 and number of trees = 50

As we can see above, each of the models performed quite well, with none of them performing alarmingly poor. The lowest F1 Score was from the Decision Tree on test data at roughly 63%. This is still a pretty great score though as it can be used to help distinguish between players deserving of a max contract or not. The Decision Tree did perform much better on its training data, and this discrepancy could be explained by the samples that ended up in the test set. Both of the Random Forest models as well as the Naïve Bayes Multiclass Classifier (both testing and training) performed in the same range of 67%-75%. While this performance is also great, the Naïve Bayes Binary Classifier and Neural Network models stand out from the others. These models’ F1 Scores range from 84%-88%, which is extremely high.

We can confidently say that our second goal has been met. These four models and their variants all succeeded in classifying NBA players into a salary range, and each allowed us to attempt to tackle this problem from a different perspective. We recognize that this is just the tip of the surface in terms of how machine learning can be used in sports, but we also recognize how powerful the models we have created are. Especially when considering the Naïve Bayes Binary Classifier and Neural Networks models, these can be confidently used by NBA agents as primary tools for stat analysis.

***Sources***

August 3rd 2021 at 7:31am CST by Luke Adams. “NBA Maximum Salaries for 2021/22.” *Hoops Rumors*, www.hoopsrumors.com/2021/08/nba-maximum-salaries-for-2021-22.html. Accessed 8 June 2023.

Papadaki, Ioanna, and Michail Tsagris. “Estimating NBA Players Salary Share According to Their Performance on Court: A Machine Learning Approach.” *arXiv.Org*, 31 Oct. 2020, arxiv.org/abs/2007.14694.