Article

Classification of NBA MVP Candidates Using Machine Learning

Matthew Schanilec [1], Robyn Halvorson [2], Isnino Harun [3], Malyun Ali [4], Matt Schoh [5] and Mohammed Mahmoud [6,*]

[1] Minnesota State University Moorhead; matthew.schanilec@go.mnstate.edu
[2] Minnesota State University Moorhead; robyn.halvorson@go.mnstate.edu
[3] Minnesota State University Moorhead; isnino.harun@go.minnesota.edu
[4] Minnesota State University Moorhead; malyun.ali@go.minnesota.edu
[5] Minnesota State University Moorhead; matt.schoh@go.mnstate.edu
[6*] Minnesota State University Moorhead; prof.mahmoud@mnstate.edu

* Correspondence: prof.mahmoud@mnstate.edu

Abstract: In recent years, the National Basketball Association (NBA) has experienced a significant increase in the use of predictive analytics by franchises and outside interest holders. As such, there is an abundance of clean, publicly available data waiting to be investigated. Additionally, each year the NBA provides an award to the Most Valuable Player (MVP), the winner of which is voted for by a panel of NBA media members. Along with this award comes great prestige, as well as significant financial incentives. The goal of this paper is to implement multiple Machine Learning (ML) algorithms on individual player statistics and MVP voting totals to accurately classify candidates in the running to win the NBA MVP award. After training and testing, accuracy testing showed very high accuracy for three algorithms: Support Vector Machines (SVM), Random Forest (RF) and Gradient Tree Boosting (GTB).

Keywords: Machine Learning, Classification, National Basketball Association, Most Valuable Player (MVP), Python.

1. Introduction

1.1. Machine Learning

As our society has shifted radically from nondigital toward the widespread, nearly universal adoption of computerized technology, many consequences have arisen. One of the consequences accompanying that shift has been an unfathomable increase in the amount of data that is created. Across every industry and all aspects of human life, an infinitude of new data points are being created. Companies like Google have constructed their entire business models around the collection, processing and distribution of data. In this paper, we will be utilizing one of the most effective tools in our new world of data: Machine Learning.

Machine Learning is a sub-field of the larger field of artificial intelligence (AI) that is concerned with creating algorithmic models that process data to complete a task. More specifically, ML involves creating and/or implementing algorithms that improve their own efficacy i.e., learn [1]. This ability to optimize is only made possible as a result of the existence of large datasets. Without any data, the entire endeavor is impossible, and without a large amount of data, any results are likely useless.

Most often, Machine Learning is implemented for its predictive qualities, although it also has strong descriptive capabilities [1]. The range of tasks in which it can be deployed are staggering, including disease prediction, chemical research, content recommendation, facial recognition, etc. [1] [2]. Indeed, given appropriate data, there are few tasks for which Machine Learning would not be fit to attempt to complete.

One of the most common types of ML tasks, and the one that is attempted in this paper, is a predictive task known as classification. Classification is an example of supervised learning in which a dataset is split into two subsets: training data and test data. The ML algorithms that are being implemented are initially given the training data as input. During this training phase, the algorithms create models that optimize themselves according to their specific method of optimization, which varies depending on the given algorithm. Then, after fully optimizing on the training data, the models are deployed on the test data to determine their predictive capabilities [3].

1.2. National Basketball Association's Most Valuable Player Award

Each season, the National Basketball Association (NBA) provides the Most Valuable Player (MVP) Award to a single player who is thought to have been the "best" player during that season. The recipient of the award is determined by a panel of NBA media members, who are each allowed to make a ranked list of five players. Each rank is assigned a different number of points. For example, for the 2020-2021 NBA season, the points per vote were assigned as follows: first place – 10 points; second place – 7 points; third place – 5 points; fourth place – 3 points; fifth place – 1 point [4].

There are multiple factors that are considered when determining whether a player is an MVP candidate. The first factor is team success. Over the last 10 seasons, the lowest team winning percentage (from 0.0 to 1.) recorded for any MVP is 0.573. The average winning percentage over that same span is 0.745 [5]. This makes it clear that voters tend to value the team success of potential MVP candidates highly. The second factor is most easily described as the relative burden placed on the candidate. For example, given the previously mentioned high team winning percentage, a candidate who is the only "star" player on their team will tend to accrue more favor than a candidate who has help from one or more teammates who are also "star" players. This is by far the most subjective factor, but it plays a large role, nonetheless. The final major factor is individual performance. Generally, the higher the level of performance, the more likely it is a candidate will receive MVP votes.

In the NBA, level of performance is readily determined by statistical measures. In the "modern" era, the NBA has seen a dramatic increase in the use of advanced metrics as well as predictive analytics. Teams use these new metrics and predictions to advise their decision-making processes. Some of the data that is gathered is private or gated by prohibitive pricing, however, much of it is publicly available. In this paper, a selection of statistics that are most likely to provide the highest predictive value are taken from these publicly available datasets to be used in tandem with a selection of ML algorithms.

2. Proposed Models

2.1. Support Vector Machines

For the purposes of this paper, a Support Vector Machine (SVM) is a type of Machine Learning model that can be used to perform classification tasks.

Generally speaking, it is capable of accomplishing other tasks, like regression and outlier detection, however, it will not be implemented in those ways for the task in the current paper [6].

SVMs fall into the larger category of supervised learning models, meaning that in order to effectively perform a given task it must be trained on a set of data that has already been assigned classes. According to Ling and Rong, SVMs have traditionally been used in instances where there are only two classes into which a sample can be assigned, otherwise known as binary classification [7]. An example of binary classification would be determining whether a lightbulb is "on" or "off". In this example, class one includes all lightbulbs that are on, while class two includes all lightbulbs that are off. SVMs also have the capability to perform classification for more than two classes [6].

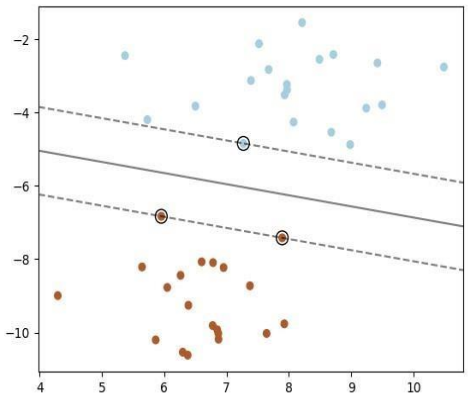

Figure 1. A graphical representation of the decision function for SVM.

During the training phase of a typical binary classification task, SVMs map data samples to points in space in accordance with the values of their attributes. After the samples have been mapped, a line – known as a hyperplane – is drawn to bisect the two classes [8]. This hyperplane is a decision boundary that determines which class each sample belongs to. In figure 1 [6], the solid line indicates the hyperplane that divides the data into two classes.

The location of the hyperplane in space is determined by points called support vectors. Support vectors are the samples from both classes that are located closest to the hyperplane in space [8]. In figure 1, the class on the bottom (red) has two support vectors, indicated by the two red dots with a dotted line running through them. The class on the top (blue) has a single support vector, indicated by the single blue dot with a dotted line running through it. The hyperplane is drawn to be parallel to and equidistant from the lines drawn through the support vectors [8].

The area in between the two support vector lines is called the margin. As the margin increases, the effectiveness of the model also increases [6]. The lines themselves are called margin boundaries. While the hyperplane is technically a decision boundary -- since any point on either side will belong to the proper class -- the margin boundaries optimize the model's ability to classify, acting as a form of secondary decision boundaries [9]. Notice that in figure 1, all samples for either class are on the opposite side of the margin boundary from the hyperplane (apart from the support vectors, which fall on the margin boundaries).

Finally, the farther a point is from the hyperplane/margin boundaries, the stronger the predictive confidence is for that sample [9].

2.2. Random Forest

Like SVMs, Random Forest (RF) algorithms fall under the larger category of supervised learning models. Again, similarly to SVMs, RFs can be utilized to perform both classification and regression tasks [10]. Using a RF algorithm can be both flexible and easy because it utilizes decision trees to discover the best solution. A decision tree is a relatively simple recursive decision-making process that can be thought of as a line that "branches" off for every attribute belonging to an input sample. RF models use large numbers of these decision trees in concert to come to predictive conclusions. Each individual decision tree's prediction is tallied as a vote, and the predictive outcome that is voted for most becomes the final predictive outcome for the entire model.

Random Forest can perform with high accuracy with all the trees involved because it takes the average of every prediction and cancels out the biases [10]. There are two important parts with the data: first, find all the data that is most crucial with each data piece. The second part involves dividing the data to find the most valuable solution, which is the most valuable player (MVP) in our scenario. The candidate that scored as the MVP is represented by 1, and the candidate that did not make the MVP is represented by 0.

From an implementation standpoint, the Python packages used in this process include pandas, scikitlearn, seaborn and matplotlib [11]. Pandas is used to create the data frame using lists for each candidates' expectations. Once the dataframe of the dataset is formed, the columns are split into dependent and independent variables.

The features are represented as X and includes the data used for each player. The labels (such as "is the MVP" or "not the MVP") are represented as y. The train_test_split process is then used in Python with scikit-learn to use model testing and training with the dataset. Then it is possible to split the inputs and outputs at the same time, with a single function call [12]. The Random Forest is then applied as shown in figure 2 [11].

```
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
```

Figure 2. Implementation of RF Classifier in Python

Once this is applied, the confusion matrix is used for actual and predicted outcomes. Below is an example with pandas seaborn and matplotlib in figure 3 [13].

Matplotlib is within Python's library and used to plot the results [14]. Setting the margins equal to True makes it easier to add the totals within the confusion matrix. Once the code is run, the confusion matrix shows the totals with a graph such that we can see all possible outcomes.

Below is an example of a prediction outcome with pandas seaborn matplotlib all in use with a confusion matrix as shows in figure 3 [13].

```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

data = {'y_Actual':    [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
        'y_Predicted': [1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0]
        }

df = pd.DataFrame(data, columns=['y_Actual','y_Predicted'])
confusion_matrix = pd.crosstab(df['y_Actual'], df['y_Predicted'], rownames=['Actual'], colnames=['Predicted

sn.heatmap(confusion_matrix, annot=True)
plt.show()
```

Figure 3. Implementation of a confusion matrix in Python.

There are several options to choose from for a good visualization graph with colors and overall shape of the data being presented. Because seaborn is built on top of matplotlib, they are both equally important to have good visualization such as what is presented below in figure 4 [14].

Figure 4. Graphical representation of a confusion matrix.

After training it is important to check the accuracy using actual predicted values. A smaller number of features reduces training time and removing the least important features causes the accuracy to increase.

Using the feature importance variable to see feature importance scores shows the relative importance or contribution of each feature.

One of the main disadvantages of using RF models is that they are expensive from a processing perspective, because of the sheer number of decision trees that need to be computed. As a result, predictive output can take more time than other, less computationally expensive methods [10].

2.3. Naïve Bayes

The Naïve Bayes (NB) algorithm dependably performs at a high level, while still being a solid option for a clean way to implement a Machine Learning algorithm [15]. The assumptions leading to the attributes of the algorithm can be a hurdle for some, which is why there has been a lot of work done on extending and morphing this algorithm.

Generally speaking, when operating in the world it can be extremely difficult for one to make assumptions with high confidence. That is part of what makes this algorithm 'naïve', in addition to the set of variables being used. The likelihood of each variable being used are independent of each other and do not take order into consideration. Simply put, NB is seeing the probability that certain variables presented are either one outcome or another. This is determined by using a training dataset.

The algorithm looks at each variable and calculates the probability of that variable being present for either outcome. Then, when presented with a

problem, the algorithm calculates the likelihood of these variables being at that specific level. To break it down even simpler, this algorithm takes the training dataset and creates histograms for each variable to find the probabilities needed for that variable to be a signal for this outcome to be chosen. In its most basic form, NB is separated into two training datasets that then trains the algorithm on the probabilities of each outcome. Then, when an outcome is presented, the algorithm makes a decision based on the likelihood it knows according to the training datasets. This technique is simple yet effective and brings value to many situations.

2.4. Gradient Tree Boosting

Gradient Tree Boosting is used to perform ensemble classification. Compared to Random Forests; Gradient Boosted Trees have a lot of model capacity, so they can model very complex relationships and model boundaries. However, more model capacity can lead to overfitting.

Random Forests use bagging to build independent decision trees whereas gradient boosted trees use a method called boosting, which combines weak learners such as stumps (a tree with only one split.) The gradient boosted tree corrects the errors of the previous one [16].

When working with regression problems, we start with a leaf that is the average value of the variable we are predicting. To calculate the average, we will add the total of all the win shares together and divide that by the number of players per year. Then, for every sample we calculated the residual [17]: Residual = actual value – predicted value.

Once we have the residuals, we are then able to construct the decision tree. Every leaf has a prediction for the value of the residual. If the residuals end up being more than the leaves, we are then overfitting, so we will have to recalculate. To prevent overfitting, there is a hyperparameter called learning rate [17] that is helpful for each prediction. The learning rate is chosen carefully. If it is too low, the model will take too long to output the right answer. In our case, we used 0.2 which is why this method is called Gradient Boosting. The prediction is then forced to be multiplied by the learning rate which causes us to use more decision-trees for the final solution. Average wins + Learning Rate * Residual predicted by decision tree

If the learning rate hypermeter is not used, the residuals will automatically share leaves, which would involve us finding these leaves and calculating the average of the 2 in one leaf. The new set of residuals is found by subtracting the actual win share from the predictions made in the previous step of constructing the tree. Every leaf will then contain a prediction for the value of the residual.

These steps are repeated until the number of iterations matches the number specified by the hyperparameter (There are no longer 2 outcomes shared in one leaf).

Once trained, we then use all the trees to make a prediction of the value of the most valuable player. The final prediction is equal to the mean computed in the first step, plus all the residuals predicted by the trees that make up the forest multiplied by the learning rate (0.2) [17]. Start by loading libraries into Python such as pandas and sklearn as shows in figure 5 [18].

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
```

Figure 5. sklearn library.

Then load in the training data using the following code in figure 6 [18].

```
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

Figure 6. Graphic of train/test data code.

Then make a concatenated new dataset by appending the data to the new set:

full_data = train_data.append(test_data)

You can also drop columns to test and see how each piece affects the output [18]:

full_data.drop(labels=drop_columns, axis1, inplace = True).

Then split the data into training sets of both x and y as shown in figure 7 [17].

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Figure 7. Graphic of split with x, y in Python.

Applying different learning rates so that there is a comparison of the performance at those rates is important to see. This is done as follows in figure 8 [18].

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=learning_rate, max_feature
    gb_clf.fit(X_train, y_train)

    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_train, y_train)))
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_val, y_val)))
```

Figure 8. Graphic of applying learning rates in Python.

The primary focus is the accuracy of the validation set. The learning rates are given to check the accuracy by inserting previous years and confirming the same output as shown in figure 9 [18].

```
Learning rate:  0.25
Accuracy score (training): 0.835
Accuracy score (validation): 0.750

Learning rate:  0.5
Accuracy score (training): 0.864
Accuracy score (validation): 0.772

Learning rate:  0.75
Accuracy score (training): 0.875
Accuracy score (validation): 0.754

Learning rate:  1
```

Figure 9. Test results with learning rates.

Then create a new classifier and specify the best learning rate found. There are faster ways to predict by using XGBoost and comparing if the results are similar. In this case, the results are very similar, but in other circumstances it is not always the case [18].

It has been proven that taking smaller incremental steps towards the solution achieves a lower variance for better accuracy on samples outside of the training data [16]. It is the overfitting process that is the most time consuming in GB.

3. Results

For each of the four ML models, 100 independent versions of the models were trained and tested on standardized, randomly split samples of the dataset [19]. Utilizing an accuracy test from the metrics model of sklearn, accuracy scores were calculated for each model version [20]. These scores were then aggregated and processed to determine the mean and median scores in the following table:

Table 1. Results.

Model	Mean Acc.	Median Acc.	Min. Acc	Max. Acc
SVM	99.14%	99.06%	97.17%	100.0%
NB	94.30%	94.34%	86.79%	100.0%
RF	99.38%	100.0%	97.17%	100.0%
GTB	99.44%	100.0%	97.17%	100.0%

Three of the four classifier models (SVM, RF, and GTB) had mean accuracy scores above 99%. Of those three models, GTB had the highest mean accuracy at 99.44%. RF and GTB shared the title for highest median accuracy at 100%.

The model with the lowest mean and median accuracy scores was NB, and although it appeared to perform at a very high level overall, with scores of 94.30% and 94.34%, respectively, its score is deceiving. Upon closer inspection, the NB model seemed to struggle with falsely classifying non-MVP candidates as MVP candidates. More specifically, it had difficulty with players whose statistical profiles were approaching MVP candidate caliber, but who had fallen short of such accomplishments. Since the MVP candidate class is so small (13 samples), relative to the non-MVP candidate class (~500), incorrectly classifying 5-6 samples as MVP candidates is a much larger problem than the accuracy score lets on at first glance.

4. Conclusion

Machine Learning is utilized across a wide range of domains for an even wider range of tasks. The present study set out to determine how effective each of four different Machine Learning models are in accurately classifying NBA players into two classes: MVP candidates and non-MVP candidates, based on individual player statistical profiles from the 2019-2020 NBA season. All in all, three of the four ML models performed their classification tasks admirably, resulting in accuracy scores of over 99%: SVM, RF, and GTB.

5. Future work

The impact of this study, on its own, is relatively small. However, as a preliminary/exploratory project, it has uncovered the potential for developing future work that could be of great value. As a result of time constraints, the scope of the project was rather limited. The following is a list of future work that should be completed in relation to the current study:

- Utilization of multiple years of individual player statistical data, allowing for a wider range of statistical profiles to be examined.
- Development of class labels that account for the number of MVP votes a player has earned. This would allow for players to be classified into stratified tiers, indicating how likely they are to win the MVP award.
- Developing a methodology to utilize ML models during the NBA season, as player statistics are constantly changing, to act as a predictor. In contrast, the current study takes advantage of a full season's worth of statistics and a list of pre-established MVP candidates.

References

1. E. Alpaydin, Introduction to Machine Learning. Cambridge: MIT Press, 2014 [Online]. Available: http://ebookcentral.proquest.com/lib/mnstate/detail.action docID=333 9851
2. Lafuente, D., Cohen, B., Fiorini, G., García, A. A., Bringas, M., Morzan, E., & Onna, D. (2021). A gentle introduction to machine learning for chemists: An undergraduate workshop using python notebooks for visualization, data processing, analysis, and modeling. Journal of Chemical Education, 98(9), 2892-2898. doi:10.1021/acs.jchemed.1c00142
3. "Introduction to Machine Learning Problem Framing." [Online].Available: https://developers.google.com/machinelearning/problemfr aming/cases [Accessed: 2021]
4. "Nikola Jokic wins 2020-21 Kia NBA Most Valuable Player Award." NBA, NBA.com, 08-Jun-2021 [Online]. Available: https://www.nba.com/news/nikola-jokic-wins-2020-21-kia-nbamostvaluable-player-award
5. "NBA MVP & ABA Most Valuable Player Award Winners," 2021. [Online]. Available: https://www.basketballreference.com/awards/mvp.html. [Accessed: 24-Oct-2021]
6. "1.4 Support Vector Machines." [Online]. Available: https://scikitlearn.org/stable/modules/svm.html#classification. [Accessed: 02-Nov-2021]
7. P. Ling and X. Rong, "A Novel and Principled Multiclass Support Vector Machine," vol. 30, no. 10, pp. 1047-1082, 2015,doi:10.1002/int.21718.[Online].Available: https://api.istex.fr/ark:/67375/WNG-R76VP03T-9/fulltext.pdf
8. State University of New York at Buffalo. (2016). Introduction to machine learning – support vector machines.[Online].Available: https://www.youtube.com/watch?v=EvMF9wvOdao
9. Udacity. Support Vector Machine – Georgia Tech – Machine Learning. (Feb. 23, 2015). Accessed Nov. 2, 2021.[OnlineVideo].Available: https://www.youtube.com/watch?v=eUfvyUeGMD8
10. A. Navlani, "Understanding Random Forests Classifiers in Python." [Online].Available:https://www.datacamp.com/community /tutorials/ra ndom-forests-classifier-python. [Accessed: 02Nov-2021]
11. "Example of Random Forest in Python," 27-Mar-2020. [Online]. Available: https://datatofish.com/random-forestpython/. [Accessed: 02-Nov-2021]
12. M. Stojiljkovic, "Split Your Dataset With scikit-learn's train_test_split()," 23-Nov-2020.[Online].Available: https://realpython.com/train-test-split-python-data/ [Accessed: 02- Nov-2021]
13. "Example of Confusion Matrix in Python," 30-Jan-2021. [Online]. Available: https://datatofish.com/confusionmatrix-python/. [Accessed: 02-Nov-2021]
14. B. Solomon, "Python Plotting With Matplotlib (Guide)," 28-Feb-2018.[Online].Available: https://realpython.com/python-matplotlib-guide/ [Accessed: 02-Nov-2021]
15. H. Langseth and T. D. Nielsen, "Latent Classification Models," vol. 59, no. 3, pp. 237–265, 2005, doi: 10.1007/s10994-005-0472-5.[Online].Available: https://search.proquest.com/docview/757010615
16. Dhingra C. "A Visual Guide to Gradient Boosted Trees (XGBoost)" 27-Dec-2020 [Online] Available: https://towardsdatascience.com/a-visual-guide-to-gradientboosted-trees-8d9ed578b33 [Accessed 10-Nov-2021]
17. C. Maklin, "Gradient Boosting Decision Tree Algorithm Explained" 17-May-2019 [Online] Available: https://towardsdatascience.com/machine-learning-part-18boosting-algorithms-gradient-boosting-in-pythonef5ae6965be4 [Accessed 10-Nov-2021]
18. Nelson D. "Gradient Boosting Classifiers in Python with Scikit-Learn" [Online] Available: https://stackabuse.com/gradient-boosting-classifiers-inpython-with-scikit-learn/ [Accessed 10-Nov-1-2021]
19. "sklearn.Preprocessing.StandardScaler." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed: 2021]
20. "sklearn.metrics.accuracy_score." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. [Accessed: 2021]