Jacob Russell

Boise State University

Final Project

Math 566: Numerical Linear Algebra

Course Instructor: Dr. Grady Wright

May 2, 2023

Introduction

Every spring, millions of basketball fans log into their ESPN accounts and create brackets for the mens' and womens' NCAA "March Madness" National Championship Tournament. This year, over 20 million people created submissions for the mens' tournament, and 2 million for the womens'. How many of those 22 million guessed all of the games perfectly, collecting Warren Buffet's $1 million prize? Absolutely zero! In fact, in the history of the tournament (nearly 85 years) nobody has ever created a perfect bracket. With 9.2 quintillion possible brackets, this fact is not overly surprising, but it does raise the question of what the best method for choosing who wins each game is.

Some people follow the advice of NCAA basketball experts and coaches while others choose based on color or which mascot would win in a fight. You would think that the experts would outperform "random guessers" by a mile, but both of these methods seem to have about the same likelihood of creating a good bracket. I was interested in researching a method of using Machine Learning (ML) to find patterns in the tournament games that a human wouldn't be capable of finding, no matter how much they knew about the NCAA, and using that to tip the odds in my favor.

When I started researching methods for predicting March Madness games I came across a contest on Kaggle.com with a $50,000 prize to the code that most accurately predicted the tournament games (Kaggle 2023). I did not have the time to create a submission for this challenge but it sparked the idea to create my own prediction technique. My first goal was to create a data set with all of the relevant variables that a basketball team accumulates throughout the season (points per game, points allowed, fouls etc.) and use the Random Forest ML method to predict this year's tournament. Then, I did a Principle Component Analysis (PCA) on that data set and ran the principle component data set through the Random Forest to see if it improved the performance of the model.

Formulation

This project began with the manipulation of the data sets for mens' NCAA basketball ranging from 2003 to 2023. Two data frames were initially constructed, one with regular season details, and the other with tournament details. The regular season data set contained thirty-four (34) variable columns with different stats that are collected throughout the season (see appendix), and row sorted by Team ID and Season. This gave each team for each season they played a list of variables that could be compared to other teams when they were matched up in the NCAA Tournament. The other data set contained the game context from each of the March Madness tournaments and simply listed Team 1, Team 1 Seed, Team 2, Team 2 Seed, Season, and Result.

BlkPerGame	PFPerGame	OppTOPerGame	OppStlPerGame	OppBlkPerGame	OppPFPerGame	Seed	Season	Result
0.766667	0.803448	-1.505747	0.827586	1.641379	-2.943678	0.0	2003.0	1.0
1.248768	1.853448	3.857143	-1.139163	-1.262315	4.140394	-15.0	2003.0	1.0
-0.827586	0.655172	0.448276	-1.275862	0.758621	2.931034	3.0	2003.0	1.0
-0.454545	3.692790	-0.991641	2.114943	-0.092999	2.295716	5.0	2003.0	1.0
-0.273563	-1.563218	0.297701	-1.502299	-0.316092	-0.229885	-1.0	2003.0	1.0
...	...	...	...	...	...	...	...	...
0.250000	1.843750	2.156250	1.843750	1.500000	1.843750	-6.0	2023.0	0.0
0.463068	-3.348485	-2.641098	-0.130682	-0.089015	-2.370265	1.0	2023.0	0.0
-1.597538	-3.714962	-0.479167	-0.697917	0.402462	-2.301136	1.0	2023.0	0.0
-1.187500	-0.656250	-0.062500	-0.687500	-0.531250	-1.468750	4.0	2023.0	0.0
-1.160038	-1.621212	-0.510417	-0.354167	0.058712	0.855114	1.0	2023.0	0.0

Figure 1: last 9 columns of the final dataset, including the results column used to calculate correct and incorrect guesses of the Random Forest

Again, this data was for every March Madness game from 2003 to 2023, the most current tournament. Then, it was a simple process of combining these two data sets, giving us a new data frame with the two teams playing each other, the difference in their seeds, the result, and the difference between each of their regular season variables (Henao 20230, Kaggle Datasets 2023).

After creating this data set, I used the Random Forest ML algorithm (Breimann 2001), using entries from 2003-2022 as training data and 2023 as the test data. This allowed me to immediately get a percentage that the algorithm would have predicted correctly from this year's tournament

without having to code a tool to create a bracket and check it. Also, by taking out 2023 data, I avoided any errors in having data exist in training that would then be used for the test.

With the results from this Random Forest, I would have predicted significantly better than average, but I wanted to make my ML model even better. With over thirty variables, there was a high risk for over-fitting and colinearity of variables. So, I decided to use the dimension reduction technique of Principle Component Analysis (PCA). PCA is a method developed by Karl Pearson in 1901 (Pearson 1901), although it was again independently developed and named by Harold Hotelling in the 1930s (Saha 2017).

PCA is one of the most widely used techniques for finding the optimal ways of combining variables into subsets, and to examine the underlying correlations of the set of data (Saha 2017). Principle component analysis is best used when there is colinearity between variables and when there are a large number of input features (Loukas 2023). As both of these are true about my data set, PCA was a .

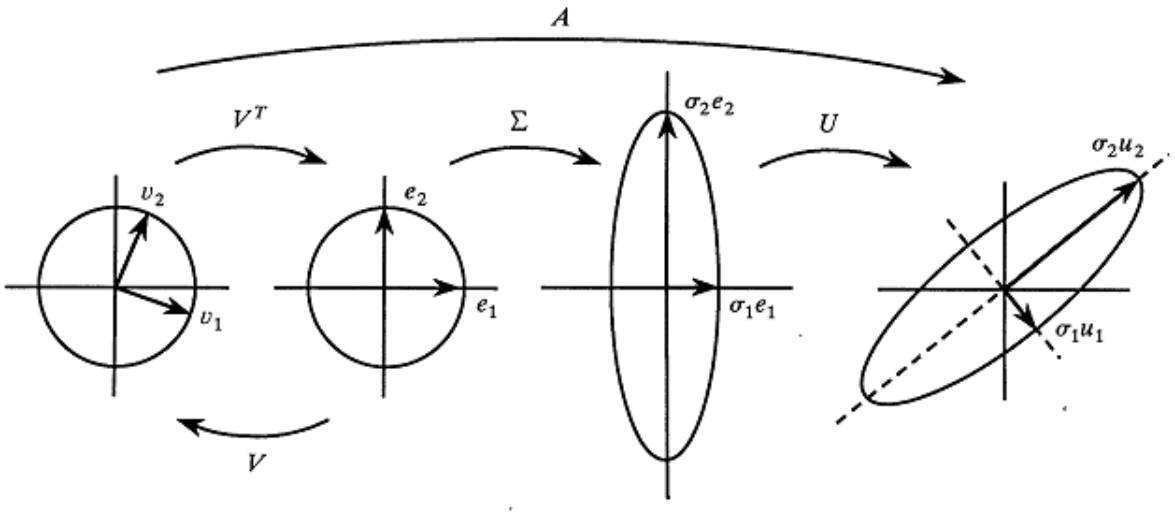

Figure 2: The transformation induced by the matrix A (the long arrow across the top of the diagram) is equivalent to the composition of the three fundamental transformations, namely a rotation, a scaling, and another rotation (Wicklin 2017)

Principle component analysis is most easily described using the Single Value Decomposition (SVD) of the matrix combined with graphical representations. "The singular value decomposition expresses any n × d matrix $\mathbf{X}$ as a product of three matrices $\mathbf{U}$, Σ, and $\mathbf{V}$" (R. and Rao 2020), and we have the SVD of $\mathbf{X}$ as:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

It is important to mention that $\mathbf{X}$ must be normalized since the PCA is the practice of changing the components to account for maximum change in variance, and if the data is not normalized, the PCA might favor variables that do not actually account for high variance (Ozel 2021). Now, with the SVD of the matrix, we have that the rows of $\mathbf{V}^T$ contain the principle directions of the matrix $\mathbf{X}$, and there are exactly as many principle directions as there were variables in the matrix (R. and Rao 2020). To reduce the dimension of $\mathbf{X}$, you remove the last $\mathbf{n}$ rows from $\mathbf{V}^T$ and then multiply $\mathbf{X} \cdot \mathbf{V}$, where $\mathbf{n}$ is however many fewer principle components you want. However, it is simpler in practice to perform the following calculation:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$
$$\mathbf{X}\cdot\mathbf{V} = \mathbf{U}\Sigma\mathbf{V}^T\cdot\mathbf{V}$$
$$\mathbf{X}\cdot\mathbf{V} = \mathbf{U}\Sigma$$

This way, instead of dropping rows from V^T and then calculating $\mathbf{X}\cdot\mathbf{V}$, you can just compute $\mathbf{U}\Sigma$ and then drop columns off until you have achieved your desired level of dimensionality (R. and Rao 2020). This is why the columns of $\mathbf{U}\Sigma$ are called the Principle Components of $\mathbf{X}$ and we are left with $\mathbf{X}$'s simplest completed Principle Component Analysis.

The majority of the computational cost of Principle Component Analysis comes during the Single Value Decomposition. Often the matrices for which principle component analysis is done are large, very dense matrices, but low-rank (Vasudevan 2019). So, the computational cost of doing the SVD step to an m × n matrix is of the order O(mn*min(m,n)) (Vasudevan 2019). The data set I created has 2,622 rows and 28 columns, so the computational cost to calculate that matrix's SVD (and essentially the PCA as well) would be (2,622*28*28) = 2,055,648 FLOPS.

Now, this does not take into account the computational costs of summing, averaging, subtracting, and combining the elements of the datasets. However, since the singular value decomposition takes up the vast majority of the computational complexity, it is a fair estimate to simply calculate it's cost as an estimate for the cost of the entire project. It is also important to remember that I did not factor in the cost of the Random Forest algorithm, or the potential cost of creating a predicted bracket; although, these computational costs are not inherently associated with Principle Component Analysis nor the SVD.

Results

For the first run of the algorithm, I manually computed and entered the results into a bracket on ESPN.com and submitted it as my official entry for the groups that I was a part of for this year's tournament. Unfortunately, that bracket resulted in a correct guess rate of 21% and a rank of 15,900,000 out of 20,056,273 brackets, leaving me in the 20th percentile of all submissions. Shortly after the tournament, I added a professional ranking system to the variables and re-ran the code, which resulted in a correct guess rate of 62.6% and a rank of 7.9 million (65th percentile).

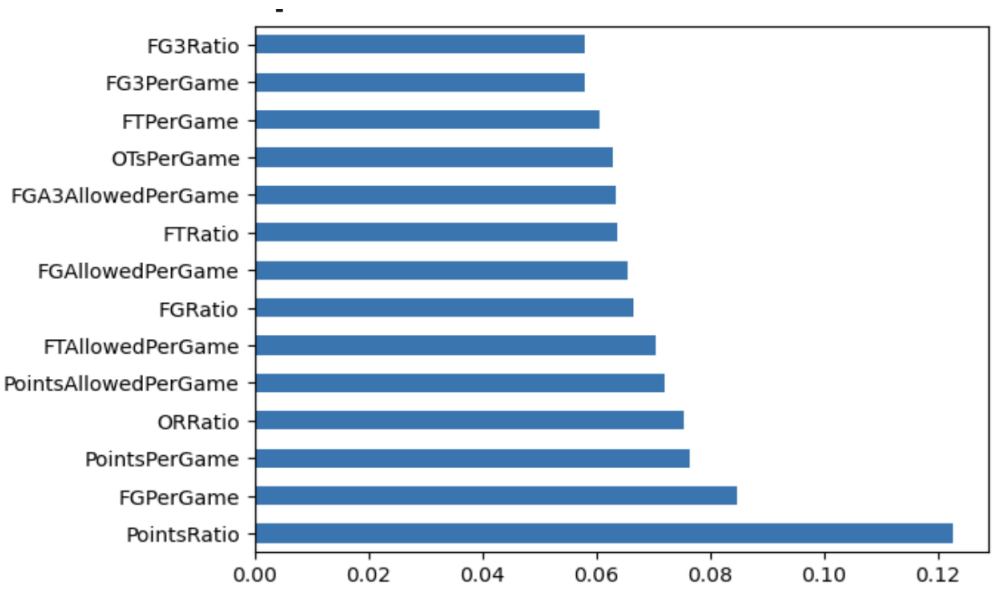

Figure 3: Top 14 variables (y axis) and the relative variance in the output of the algorithm (x axis)

However, when I plotted the most important variables, that new professional ranking system

was not one of the top 10 most important. Since there was a 40% jump from the first run to the second, I think that I must have made an error in calculating and entering the results. If this jump was caused by the new variable, then that variable would have ranked very high in importance. Since it did not, I can only assume that something else went wrong between training and testing the first model, and entering its results for the 2023 tournament on ESPN.com.

	Cumulative Variance Ratio	Explained Variance Ratio
0	0.197145	0.197145
1	0.388145	0.191001
2	0.510047	0.121902
3	0.595391	0.085344
4	0.665475	0.070084
5	0.717137	0.051662
6	0.755698	0.038561
7	0.791969	0.036271
8	0.826013	0.034044
9	0.858338	0.032325
10	0.886649	0.028311
11	0.912978	0.026329
12	0.931159	0.018181
13	0.946662	0.015503
14	0.959533	0.012871
15	0.971745	0.012212
16	0.979743	0.007997
17	0.986261	0.006519
18	0.991137	0.004876
19	0.995091	0.003954

Figure 4: Cumulative variance ratio and explained variance ratio by principle components

The next step was to do a principle component analysis on the dataset, and run that PCA dataset through the Random Forest. First I had to standardize and normalize the data (see Appendix) and then I used the built-in PCA tool in python from the sklearn.decomposition package (Scikit). This yielded a new data set, also with 28 columns, but with each column being a principle component.

By plotting the cumulative variance explained by the principle components, I could see that 10 principle components accounted for nearly 90% of the variance, and 95% was explained by 13 principle components.

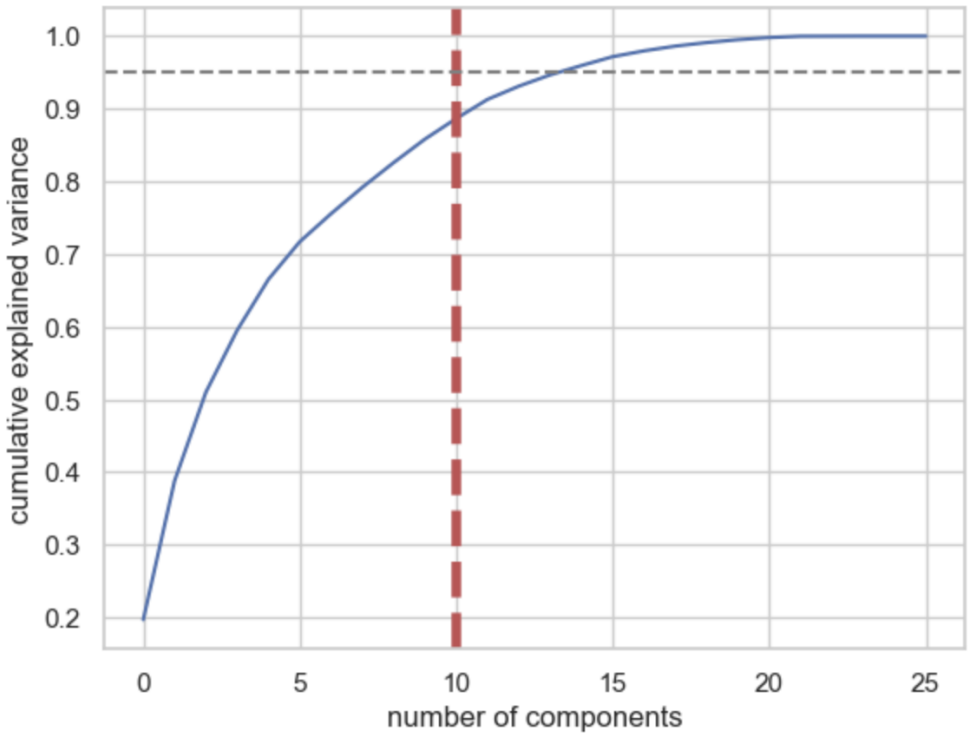

Figure 5: Cumulative explained variance plot of principle components

By running this dataset through the Random Forest multiple times, I found that the most accurate results occurred while using 18 principle components.

Those 18, when run through the Random Forest, correctly predicted 75.4% of the March Madness games from the 2023 tournament. This resulted in a rank of 4.7 million and placed in the 77th percentile of brackets on ESPN.com. This is a clear improvement from 21% on the first try and 62.3% on the second. Although 75% would not have won any of the bracket challenges I entered, I still see the PCA as a win because of the substantial improvement it made on the model.

Conclusion

This project focused on building a ML algorithm to predict March Madness games from 2023 as accurately as possible. A dataset consisting of over 30 basketball variables for each NCAA national championship tournament game since 2023 was used in a Random Forest algorithm, resulting in a slightly above average guess percentage. Then, through the Principle Component Analysis of the

original data, a new data frame was created, which resulted in marked improvement on the model.

I learned a multitude of things throughout this project. Gathering and cleaning data as well as building a useful data set is a skill-set, and a very time intensive process. I have done a number of projects which required data manipulation, but never on my own, and never as involved and intense as this. The ML and statistical analysis of data is only as valuable as what is going in, and I really came to terms with this fact during this work. I also learned about the process of Principle Component Analysis. We learned about the basics of the technique during class, but through this research and doing it on a real set of data, my understanding of the process grew and I feel much more comfortable with it than I did before.

The other major project I worked on this semester also involved Random Forests, so it was nice to see the similarities and differences between the two. The other project involved climate and topographical variables, while this had everything to do with sports; it was quite interesting to see how two very different concepts can utilize the same techniques to predict outcomes. I think that there is a lot more for me to learn in the realm of ML, specifically other methods. Also, I know I have a lot to learn about data cleaning and manipulation and how to take advantage of the structure of whatever data set I am working on.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10
0	0.886205	-1.874887	1.987176	-0.088023	-0.939516	1.463554	-0.703984	0.104257	-1.101833	-0.263543
1	3.716335	3.494558	0.453975	-1.625686	1.647612	1.329644	0.100193	0.617819	-1.096467	0.856785
2	1.040925	-0.959595	-0.140176	1.912052	1.438175	0.510046	1.363232	1.134300	0.208196	0.150797
3	2.725300	-2.618798	0.974281	1.057772	2.075510	-1.043366	-0.205012	-1.192580	-0.948524	1.148824
4	-0.024690	1.038982	-0.145920	1.789095	-1.329070	0.113679	0.962226	1.716190	0.451124	-0.720747
...	...	...	...	...	...	...	...	...	...	...
2491	-1.401312	-1.069966	1.971521	-0.985079	-1.569531	1.058754	2.298490	-0.730702	0.398682	0.202706
2492	0.564170	-3.936054	-4.524120	-2.019645	0.262863	-0.122558	-0.420936	0.028174	0.231628	0.241081
2493	-2.935014	-0.550540	1.157940	-1.470379	1.277482	1.609673	0.225198	-1.414141	0.739152	-0.246606
2494	0.164853	2.137327	-1.383441	-0.776783	-1.465579	0.661704	-0.174438	0.270200	-0.345727	0.928559
2495	-1.283527	-0.530918	2.285952	1.015085	0.248797	-0.198307	0.200254	-0.750445	0.986381	-0.424297

2496 rows × 18 columns

Figure 6: PCA data set with first 10 principle component columns

Principle Component Analysis is a very powerful statistical tool which can yield incredible

results. However, it does have it's drawbacks. One that I saw quite clearly was that when a PCA is run on a data set, the context of the actual variables is lost. While my original dataset had variables like Points Per Game, Seed Difference, Turnover Ratio etc., the PCA dataset simply had PC1, PC2, and so on. It did yield much better results, but a basketball fan would have no idea what specific things a team might be doing that increase or decrease their odds of winning. Also, when I plotted the first five principle components against one another, none of the plots seemed to have any patterns whatsoever.

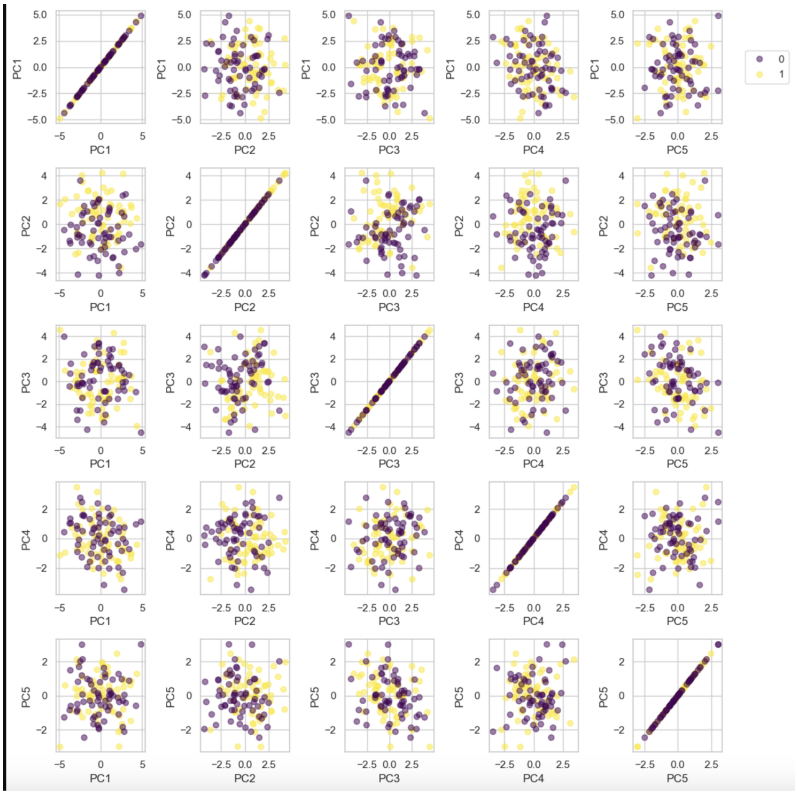

Figure 7: First 5 principle component scatterplots

One would expect some distribution pattern between principle components, and the fact that there seemingly isn't any could indicate that the principal components are not capturing meaningful patterns or variability in the data. This is not always the case, and certainly the marked improvements suggests otherwise, but it is possible that PCA might not be the right or the best option to capture relevant information from this data.

Overall this project was an incredible learning opportunity for me. I was able to couple my love

of basketball with my intense interest in predictive models and saw meaningful improvements in my knowledge and the output of the program. Given the opportunity, I would love to dive deeper and learn more about these March Madness predictions, and I definitely think that this project will help me in my career and future as a data scientist and machine learning engineer. My model might have put me far outside the realm of contention for this year's March Madness, but I fully intend to make time next spring to refine my techniques and give myself a better chance of creating the very first ever perfect bracket.

References

[1] Breiman, Leo. “Random Forests.” Machine Learning, vol. 45, no. 1, 2001, pp. 5–32., https://doi.org/10.1023/a:1010933404324.

[2] Henao, Wilmer E. “Ken Pomeroy’s Data Updated for the 2023 March Madness Season.”

[3] Https://Www.kaggle.com/Datasets/Verracodeguacas/kenpom2023.

[4] Kaggle DataSets. “March Machine Learning Mania 2023: Data Sets.” Https://Www.kaggle.com/Competitions/March-Machine-Learning-Mania-2023/Data.

[5] Loukas, Serafeim. ”PCA CLearly Explained How, When, WHy to Use It and Feature Important: A Guide in Python.” Medium, Towards Data Science, 21 Apr. 2023.

[6] Ozel, Sera Giz. “The Math behind: Everything about Principle Component Analysis (PCA).” Medium, Geek Culture, 26 Apr. 2021, https://medium.com/geekculture/the-math-behind-everything-about-principle-component-analysis-pca-d6f0baff5681.

[7] Pearson, Karl. “On Lines and Planes of Closest Fit to Systems of Points in Space.” The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 2, no. 11, 1901, pp. 559–572., https://doi.org/10.1080/14786440109462720.

[8] R., Srinivasa Rao Arni S, and C. Radhakrishna Rao. “Dimensionality Reduction and PCA.” Principles and Techniques for Data Science, North-Holland, Amsterdam, The Netherlands, 2020.

[9] Rustyb. “March Machine Learning Mania 2023.” Kaggle, https://www.kaggle.com/competitions/march-machine-learning-mania-2023/discussion/399553.

[10] Saha, Sumanta, and Sharmistha Bhattacharya Halder. “A SURVEY: PRINCIPAL COMPONENT ANALYSIS (PCA).” International Journal of Advance Research in Engineering, Science & Technology, vol. 6, no. 6, June 2017, pp. 312–329., https://doi.org/10.26527/ijarest.

[11] “SKLEARN.DECOMPOSITION.PCA.” Scikit, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[12] Vasudevan, Vinita, and M. Ramakrishna. “A Hierarchical Singular Value Decomposition

Algorithm for Low Rank Matrices." Cornell University

[13] ArcXiv Distribution Service, 10 May 2019, https://doi.org/10.48550/arXiv.1710.02812.

[14] Wicklin, Rick. "The Singular Value Decomposition: A Fundamental Technique in Multivariate Data Analysis." The DO Loop, 28 Aug. 2017, https://blogs.sas.com/content/iml/2017/08/28/singular-value-decomposition-svd-sas.html.

[15] "Your Machine Learning and Data Science Community." Kaggle, https://www.kaggle.com/.

Appendix

```
[Jupyter Lab, Python]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#Pulling in the data sets we need
folder = 'Data/Mens Data/'
Seeds = pd.read_csv(folder+'MNCAATourneySeeds.csv')
MasseyOrdinals = pd.read_csv(folder+'MMasseyOrdinals.csv')
TourneyCompact = pd.read_csv(folder+'MNCAATourneyCompactResults.csv')
RegularDetail = pd.read_csv(folder+'MRegularSeasonDetailedResults.csv')
#Massey Ordinals Cleaning
#drop Ranking Day Number, we'll just average for a season
#Also averaging all of the ordinals, so system name (name of the measurement)
is dropped as well
MasseyOrdinals = MasseyOrdinals.drop(['RankingDayNum','SystemName'],axis=1)
MasseyOrdinals = MasseyOrdinals.groupby(['Season','TeamID']).last().reset_index()
#grouping by these three and averaging the
#OrdinalRank for the entire season for a team
display(MasseyOrdinals)
#Building a winners data frame and a losers data frame
WinTeams = pd.DataFrame()
LoseTeams = pd.DataFrame()
columns = ['Season', 'WTeamID','LTeamID', 'Points','OppPoints',
       'Loc', 'NumOT', 'FGM', 'FGA', 'FGM3', 'FGA3', 'FTM', 'FTA',
       'OR', 'DR', 'Ast', 'TO', 'Stl', 'Blk', 'PF', 'OppFGM', 'OppFGA',
       'OppFGM3', 'OppFGA3', 'OppFTM', 'OppFTA', 'OppOR', 'OppDR', 'OppAst', 'OppTO',
       'OppStl', 'OppBlk', 'OppPF']
WinTeams[columns] = RegularDetail[['Season','WTeamID','LTeamID', 'WScore', 'LScore',
       'WLoc', 'NumOT', 'WFGM', 'WFGA', 'WFGM3', 'WFGA3', 'WFTM', 'WFTA',
       'WOR', 'WDR', 'WAst', 'WTO', 'WStl', 'WBlk', 'WPF', 'LFGM', 'LFGA',
       'LFGM3', 'LFGA3', 'LFTM', 'LFTA', 'LOR', 'LDR', 'LAst', 'LTO',
       'LStl', 'LBlk', 'LPF']]
WinTeams['Wins'] = 1
WinTeams['Losses'] = 0
LoseTeams[columns] = RegularDetail[['Season','LTeamID','WTeamID', 'LScore', 'WScore',
       'WLoc', 'NumOT', 'LFGM', 'LFGA', 'LFGM3', 'LFGA3', 'LFTM', 'LFTA',
       'LOR', 'LDR', 'LAst', 'LTO', 'LStl', 'LBlk', 'LPF', 'WFGM', 'WFGA',
       'WFGM3', 'WFGA3', 'WFTM', 'WFTA', 'WOR', 'WDR', 'WAst', 'WTO',
       'WStl', 'WBlk', 'WPF']]
def change_loc(loc):
    if loc == 'H':
        return 'A'
    elif loc == 'A':
```

```
        return 'H'
    else:
        return 'N'
LoseTeams['Loc'] = LoseTeams['Loc'].apply(change_loc)
LoseTeams['Wins'] = 0
LoseTeams['Losses'] = 1
#Concatanating the winning and losing data frames
WinLoseTeams = pd.concat([WinTeams,LoseTeams])
#and then adding in the ordinals for the winners and losers
WinLoseTeams = WinLoseTeams.merge(MasseyOrdinals,left_on=['Season','WTeamID'],
right_on=['Season','TeamID']).merge(MasseyOrdinals,left_on=['Season','LTeamID'],
right_on=['Season','TeamID'])
display(WinLoseTeams)
#Dropping columns
WinLoseTeams = WinLoseTeams.drop(['TeamID_x','TeamID_y','LTeamID'],axis=1)
WinLoseTeams = WinLoseTeams.rename(columns={'WTeamID':'TeamID',
'OrdinalRank_x':'OrdinalRank','OrdinalRank_y':
'OppOrdinalRank'})
WinLoseTeams = WinLoseTeams[WinLoseTeams['Season']>=2003].reset_index(drop=True)
display(WinLoseTeams)
#aggregating by team and season
combinedTeams = WinLoseTeams.groupby(['Season','TeamID']).agg({
    'Points': 'sum','OppPoints': 'sum','Wins': 'sum','Losses',
'sum','NumOT': 'sum','FGM': 'sum','FGA': 'sum','FGM3':'sum',
'FGA3': 'sum','FTM': 'sum','FTA': 'sum','OR': 'sum','DR': 'sum',
'Ast': 'sum','TO': 'sum','Stl': 'sum','Blk': 'sum','PF': 'sum',
'OppFGM':'sum','OppFGA': 'sum','OppFGM3': 'sum','OppFGA3': 'sum',
'OppFTM': 'sum','OppFTA': 'sum','OppOR': 'sum','OppDR': 'sum',
'OppAst': 'sum','OppTO': 'sum','OppStl': 'sum','OppBlk': 'sum',
'OppPF': 'sum','OrdinalRank':'mean','OppOrdinalRank':'mean'
})
combinedTeams['NumGames'] = combinedTeams['Wins']+combinedTeams['Losses']
#combinedTeams = combinedTeams.reset_index()
display(combinedTeams)
#creating regular season averages for each season for each team
RegularSeasonInput = pd.DataFrame()
#RegularSeasonInput['Season'] = combinedTeams['Season']
#RegularSeasonInput['TeamID'] = combinedTeams['TeamID']
RegularSeasonInput['WinRatio'] = combinedTeams['Wins']/combinedTeams['NumGames']
RegularSeasonInput['PointsPerGame'] = combinedTeams['Points']/combinedTeams['NumGames']
RegularSeasonInput['PointsAllowedPerGame'] =
combinedTeams['OppPoints']/combinedTeams['NumGames']
RegularSeasonInput['PointsRatio'] = combinedTeams['Points']/combinedTeams['OppPoints']
RegularSeasonInput['OTsPerGame'] = combinedTeams['NumOT']/combinedTeams['NumGames']
RegularSeasonInput['FGPerGame'] = combinedTeams['FGM']/combinedTeams['NumGames']
```

```
RegularSeasonInput['FGRatio'] = combinedTeams['FGM']/combinedTeams['FGA']
RegularSeasonInput['FGAllowedPerGame'] =
combinedTeams['OppFGM']/combinedTeams['NumGames']
RegularSeasonInput['FG3PerGame'] = combinedTeams['FGM3']/combinedTeams['NumGames']
RegularSeasonInput['FG3Ratio'] = combinedTeams['FGM3']/combinedTeams['FGA']
RegularSeasonInput['FGA3AllowedPerGame'] =
combinedTeams['OppFGM3']/combinedTeams['NumGames']
RegularSeasonInput['FTPerGame'] = combinedTeams['FTM']/combinedTeams['NumGames']
RegularSeasonInput['FTRatio'] = combinedTeams['FTM']/combinedTeams['FTA']
RegularSeasonInput['FTAllowedPerGame'] =
combinedTeams['OppFTM']/combinedTeams['NumGames']
RegularSeasonInput['ORRatio'] =
combinedTeams['OR']/(combinedTeams['OR']+combinedTeams['OppDR'])
RegularSeasonInput['DRRatio'] =
combinedTeams['DR']/(combinedTeams['DR']+combinedTeams['OppOR'])
RegularSeasonInput['AstPerGame'] =
combinedTeams['Ast']/combinedTeams['NumGames']
RegularSeasonInput['TOPerGame'] = combinedTeams['TO']/combinedTeams['NumGames']
RegularSeasonInput['StlPerGame'] =
combinedTeams['Stl']/combinedTeams['NumGames']
RegularSeasonInput['BlkPerGame'] =
combinedTeams['Blk']/combinedTeams['NumGames']
RegularSeasonInput['PFPerGame'] = combinedTeams['PF']/combinedTeams['NumGames']
RegularSeasonInput['OppTOPerGame'] = combinedTeams['OppTO']/combinedTeams['NumGames']
RegularSeasonInput['OppStlPerGame'] = combinedTeams['OppStl']/combinedTeams['NumGames']
RegularSeasonInput['OppBlkPerGame'] = combinedTeams['OppBlk']/combinedTeams['NumGames']
RegularSeasonInput['OppPFPerGame'] = combinedTeams['OppPF']/combinedTeams['NumGames']
display(RegularSeasonInput)
#each game from 2003-2023 NCAA tournaments
seed_dict = Seeds.set_index(['Season', 'TeamID'])
TourneyInput = pd.DataFrame()
winIDs = TourneyCompact['WTeamID']
loseIDs = TourneyCompact['LTeamID']
season = TourneyCompact['Season']
winners = pd.DataFrame()
winners[['Season','Team1','Team2']] = TourneyCompact[['Season','WTeamID','LTeamID']]
winners['Result'] = 1
losers = pd.DataFrame()
losers[['Season','Team1','Team2']] = TourneyCompact[['Season','LTeamID','WTeamID']]
losers['Result'] = 0
TourneyInput = pd.concat([winners,losers])
TourneyInput = TourneyInput[TourneyInput['Season']>=2003].reset_index(drop=True)
team1seeds = []
team2seeds = []
for x in range(len(TourneyInput)):
```

```
    idx = (TourneyInput['Season'][x], TourneyInput['Team1'][x])
    seed = seed_dict.loc[idx].values[0]
    if len(seed) == 4:
        seed = int(seed[1:-1])
    else:
        seed = int(seed[1:])
    team1seeds.append(seed)

    idx = (TourneyInput['Season'][x], TourneyInput['Team2'][x])
    seed = seed_dict.loc[idx].values[0]
    if len(seed) == 4:
        seed = int(seed[1:-1])
    else:
        seed = int(seed[1:])
    team2seeds.append(seed)
TourneyInput['Team1Seed'] = team1seeds
TourneyInput['Team2Seed'] = team2seeds
display(TourneyInput)
#adding in season averages context to the tournament games
outscores = []
for x in range(len(TourneyInput)):
    idx = (TourneyInput['Season'][x], TourneyInput['Team1'][x])
    team1score = RegularSeasonInput.loc[idx]
    team1score['Seed'] = TourneyInput['Team1Seed'][x]
    idx = (TourneyInput['Season'][x], TourneyInput['Team2'][x])
    team2score = RegularSeasonInput.loc[idx]
    team2score['Seed'] = TourneyInput['Team2Seed'][x]
    outscore = team1score - team2score
    outscore['Season'] = TourneyInput['Season'][x]
    outscore['Result'] = TourneyInput['Result'][x]
    outscores.append(outscore)
outscores = pd.DataFrame(outscores)
display(outscores)
import pandas as pd
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_outscores = outscores[outscores["Season"] != 2023]
# Training data includes all seasons except 2023
test_outscores = outscores[outscores["Season"] == 2023]
# Testing data includes only the 2023 season
train_outscores = train_outscores.drop(["Season"],axis = 1)
test_outscores = test_outscores.drop(["Season"],axis = 1)
# Split the training and testing data into features (X) and labels (y)
X_train = train_outscores.drop(["Result"], axis=1)
y_train = train_outscores["Result"]
```

```
X_test = test_outscores.drop(["Result"], axis=1)
y_test = test_outscores["Result"]
import numpy as np
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)
y_train = np.array(y_train)
# Train the model using the training data
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
# Test your model using the testing data
y_pred = rf.predict(X_test_scaled)
import pandas as pd
import numpy as np
data = train_outscores
X = data.iloc[:,1:15]  #independent columns
y = data.iloc[:,-1]    #target column
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of
tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
pca_test = PCA(n_components=26)
pca_test.fit(X_train_scaled)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)
plt.axhline(y=0.95, color='grey', linestyle='--')
display(plt.show())
evr = pca_test.explained_variance_ratio_
cvr = np.cumsum(pca_test.explained_variance_ratio_)
pca_df = pd.DataFrame()
pca_df['Cumulative Variance Ratio'] = cvr
```

```
pca_df['Explained Variance Ratio'] = evr
display(pca_df.head(20))
#PCA Step
pca = PCA(n_components=18)
pca.fit(X_train_scaled)
X_train_scaled_pca = pca.transform(X_train_scaled)
X_test_scaled_pca = pca.transform(X_test_scaled)
import numpy as np
import matplotlib.pyplot as plt
# Assuming X_pca is a 2D array of shape (n_samples, n_components)
# with the first two principal components being in columns 0 and 1
x = X_train_scaled_pca[:, 0]
y = X_train_scaled_pca[:, 1]
plt.scatter(x, y, c=y_train, cmap='viridis', alpha=0.5)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
#to look at the relationships between the first 5 principle components
# Assuming X_train is a 2D array of shape (n_samples, n_features)
X_train2 = X_train_scaled_pca
# Instantiate a PCA object with n_components=5
pca = PCA(n_components=5)
# Fit the PCA model to the data and transform it
X_train_scaled_pca2 = pca.fit_transform(X_train2)
n_components = X_train_scaled_pca2.shape[1]
fig, axs = plt.subplots(n_components, n_components, figsize=(12, 12))
for i in range(n_components):
    for j in range(n_components):
        ax = axs[i, j]
        if i == j:
            scatter = ax.scatter(X_test_scaled_pca[:, j], X_test_scaled_pca[:, i],
                      c=y_test, cmap='viridis', alpha=0.5)
            ax.set_xlabel('PC{}'.format(j+1))
            ax.set_ylabel('PC{}'.format(i+1))
        elif j < i:
            scatter = ax.scatter(X_test_scaled_pca[:, j],
            X_test_scaled_pca[:, i], c=y_test,
                      cmap='viridis', alpha=0.5)
            ax.set_xlabel('PC{}'.format(j+1))
            ax.set_ylabel('PC{}'.format(i+1))
        else:
            scatter = ax.scatter(X_test_scaled_pca[:, j],
            X_test_scaled_pca[:, i], c=y_test, cmap='viridis', alpha=0.5)
            ax.set_xlabel('PC{}'.format(j+1))
            ax.set_ylabel('PC{}'.format(i+1))
```

```
        if i == 0 and j == n_components-1:
            handles, labels = scatter.legend_elements()
            legend = ax.legend(handles, labels, loc='center',
            bbox_to_anchor=(1.5, 0.5))
            fig.suptitle(legend.get_label(), fontsize=25)
    for k in range(i+1, n_components):
        axs[k, i].axis('on') # don't show redundant plots
fig.tight_layout()
plt.show()
#results of random forest on PCA data set
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=1)
model = model.fit(X_train_scaled_pca,y_train)
model.score(X_test_scaled_pca,y_test)
#results of random forest on regular data set
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=1)
model = model.fit(X_train,y_train)
model.score(X_test,y_test)
```