NBA Seeds with Graph Neural Networks

Steven Liu

Aurelio Barrios

Abstract:

The NBA contains many challenges when attempting to make predictions. The performance of a team in the NBA is difficult because many things can happen over the course of 81 games. Our analysis attempts to produce accurate results by exploiting the natural structure of the NBA league and data of previous player stats. Our analysis begins with identifying the players on each roster to create an aggregated stat for each team, then we will take advantage of the schedules of each team to learn the unique performance of a team against every other team. Leveraging the features and the schedule of the teams, we expect to be able to make decent predictions of NBA seedings before a season starts.

Introduction:

The NBA is one of the most popular sports in the U.S. It is the most followed sports league on social media with more than 150 million followers. With this many people keeping track of sport, we should expect that statistics on the sport will be useful for many. An example is that we can apply our results from this project into making sports bets. If we develop a model that can produce results with good accuracy, then we can potentially use this model to place winning bets on the sport. Our assumption is that the schedule of a team matters on their rankings in the

season, this could make the NBA management aware that the schedule is making the determination of the seedings or likelihood of winning the competition unfair for some teams. This hasn't been something that has been addressed in U.S. sports yet despite similar concerns from fans and casters. It is common knowledge that the 'Eastern Conference' has always contained weaker teams compared to the 'Western Conference' which made achievements from the 'Eastern Conference' less acknowledged. Hopefully, we can make it more obvious that some of the ways the league conducts itself is unfair.

The data we decided to use for our model will simply be player stats, team rosters, and team schedules. With team rosters, we will be able to determine the aggregate team stats with the player stats, and knowing the match ups between the teams, we will be able to get an understanding of how difficult the season will be for each team. The relationship between teams will be represented with an edge, and each team will be a node. Each node will contain an aggregation of player stats, and we will have a fully developed graph network to input into our graph neural network model.

We will be implementing two models to predict the seedlings of the teams. We will apply the classical Graph Convolutional Network, GCN and GraphSAGE to our network. With these models we will be able to perform experiments such as moving players around the teams and observing the change in seedings.

Prior Work:

FiveThiryEight, a popular analytics website, developed their own NBA Ranking Prediction model. The model they developed is called RAPTOR

However, their model only looks at individual players stats and their projected stats in the future with similar NBA players. It completely ignores the record of a team, and the difference in performance they would have across the different teams they would have to face. NBA seedings predictions are heavily influenced by the performance in the current season, but this also makes predictions less impressive. In our work, we are able to take advantage of the large amount of data from previous NBA seasons starting from 1946. This allows our model to have a plethora of data to learn from which will allow us to make accurate predictions without looking at the statistics from the season we want to predict.

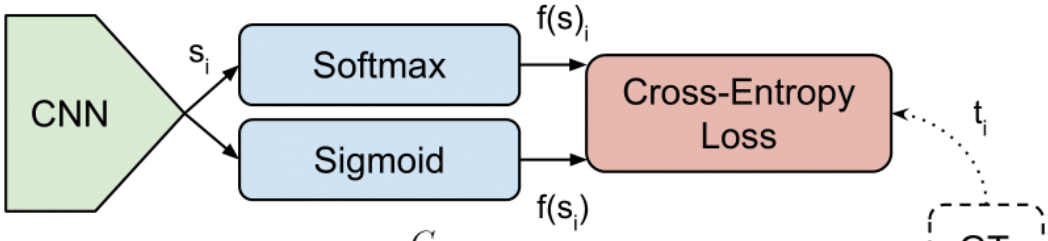
Methods:

When implementing the traditional GCN, the GCN model is initialized with the number of nodes, n, number of hidden layers, l, and the number of classes, c. Inside the GCN model, we have the GCN layers which are initialized with an adjacency matrix of structure of the graph, A and the feature matrix, X. The shape of the adjacency matrix should be n x n. Thus the feature matrix would be n x f, where f is the number of features. We will use ReLU as the activation function, σ, of these layers and use a softmax to make the classification. To train and test the model, we will need to call the forward method of the GCN model, which will take

an adjacency matrix of node edges, and a feature matrix. There is a parameter in the GCN forward method to specify whether you want to use Kipf & Welling's normalization of the adjacency matrix, A, or to leave it unnormalized.

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right) \qquad f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

The left formula shows how the layer will be computed without normalizing A, while the right shows how it will be computed with normalizing A. The model can use cross entropy loss to tune the weight parameters in each GCN layer. In our case, we will use a categorical cross entropy loss.

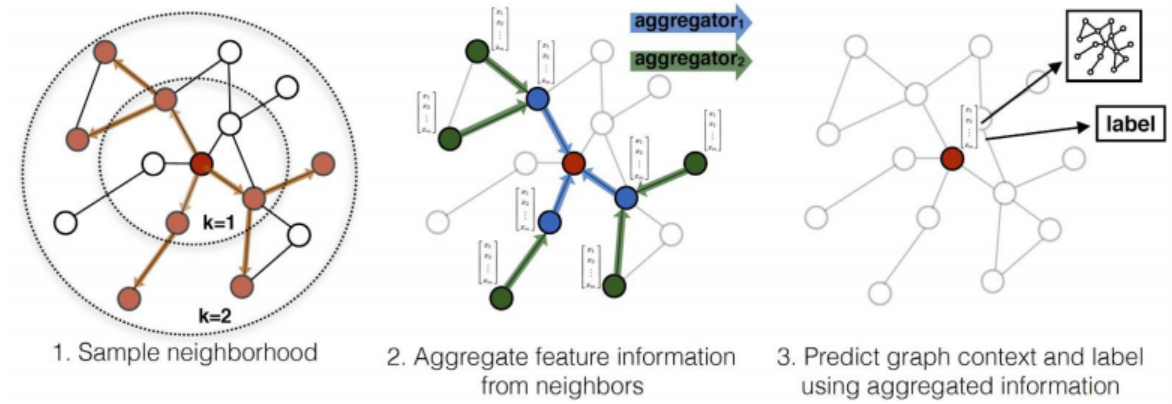


$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

GraphSAGE will create batches of neighborhoods and aggregate the information from these neighborhoods into a new feature matrix. The forward method will take in a batch of nodes, b and the depth size, k to develop a neighborhood. The model will then use b and k to create a subsample of A, an adjacency matrix that defines

the neighborhood of every node in batch, b.



The batch of nodes, b and subsample of A will be then used as input to the GraphSAGE aggregators, mean and pooling that can also be specified as a parameter in the forward method.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

The mean aggregator layer will output a feature matrix that contains the average of neighborhood features. The pooling aggregator layer will output the original feature matrix b, concatenated with the pooling of the neighborhood for

each node in b. The forward algorithm here should return a feature matrix for each node. In order to make the algorithm more efficient, we can use matrix multiplication instead of a for loop in line 3.We will multiply the matrix of the last iteration, h, with A. This results in a matrix that contains the sum of the features of every node. We can tweak this updated matrix depending on which aggregator we decide to use. What makes the GraphSAGE loss function interesting is that it only requires the output of the GraphSAGE model as input.

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

The loss function is calculated based on the similarity between the input, thus it doesn't need to know the true label of a node. It is also possible to use stochastic gradient descent as the loss function. The parameters that will be tuned are the weights of the aggregator layers.

We will be testing the performance of the GraphSAGE model with it's different aggregators, Mean, MeanPooling, maxPooling and Sequential. We will compare the performance on the test set we get for each aggregator by developing plots of Loss vs Epochs for each of the aggregators. With this information, we should be able to determine one aggregator to use for the rest of the report.

To obtain our data, we used some third-party code to scrape the basketball-reference.com website for player statistics, team rosters, and team schedules. We had to modify some of the code in order to get the data cleanly. Then

with this code, we are able to obtain data from whatever season or team listed in the basketball-reference website which contains a large amount of data.

After scraping the data from the website, we need to reformat the data into a compatible format for our models. Some problems with the data is that there was the TOT team which consisted of the overall stats of players that were traded, due to our time constraint we weren't able to incorporate this data into our data even though we are sure it would've been helpful.

In order for our web-scraped data to be useful for our model we needed to change the data from individual statistics to team statistics. In order to do this we had to aggregate all the statistics of players in their respective teams. In order to do this we had to find a way to include the data of a team as a whole, while also preserving the data of the individual players. We did this by taking into account all possible combinations of data for each team. For example, each player had a points (pts) statistic in their original data format. In order to take into account this statistic at a team level we averaged all the points for every player in a team. This is fairly straightforward, but what we also did was take into account the minimum and maximum values of points in the team while also taking note of the standard deviation. This way our data takes into account the team as a whole and if there are any notable statistics, maybe a team had a player who scored significantly more points than the rest of his team, we can also take those into account. We did this across all the features originally found and ended with 184 features for each team in

the NBA. After formatting the data, we saved them as csv files to use as input to our models which can be specified for faster results as scraping and formatting the data will take some time.

We realized that when predicting seeds, we don't need to penalize the model too harshly when making predictions that are wrong, but close. To encourage this, we decided to use binary labels instead of the actual rankings, this way the model would be able to take the advantage of its loss function better. The binary label would be whether the team made it or did not make it to the playoffs. However, we also need to modify our GraphSAGE model to output probabilities instead of a label, this is because we will use the probabilities of each label to determine the rankings of each NBA team. We will rank the teams based on their probability of making it to the playoffs. This method will increase the accuracy because it gives the model more data to work with for each label.

For our model, we need to decide how many seasons or years of the NBA will belong in our training set. Understanding sports, we know that if we include too many years, then this will cause the results to be inaccurate as players do tend to age and you shouldn't expect a player from 20 years ago to perform just as well today. This idea applies the same to teams, and generally you can see a shift in the best teams change around every 5 years. We will be testing the data with training sets from the last 1 to 10 seasons of the NBA, and whatever seasons that aren't used will be put into the validation set. Based on the results of this, we will be able to

determine the proportion of seasons to use in the training and validation sets of the model.

Results:

This section will contain the results of predicting the NBA Ranking of 2019, with data from 2011-2018.

Model Accuracy Comparison

Model	**Test Accuracies**
GCN	**53.33%**
GraphSAGE Mean	**80.00%**
GraphSAGE MeanPooling	**70.00%**
GraphSAGE MaxPooling	**76.67%**
GraphSAGE Sequential	**73.33%**

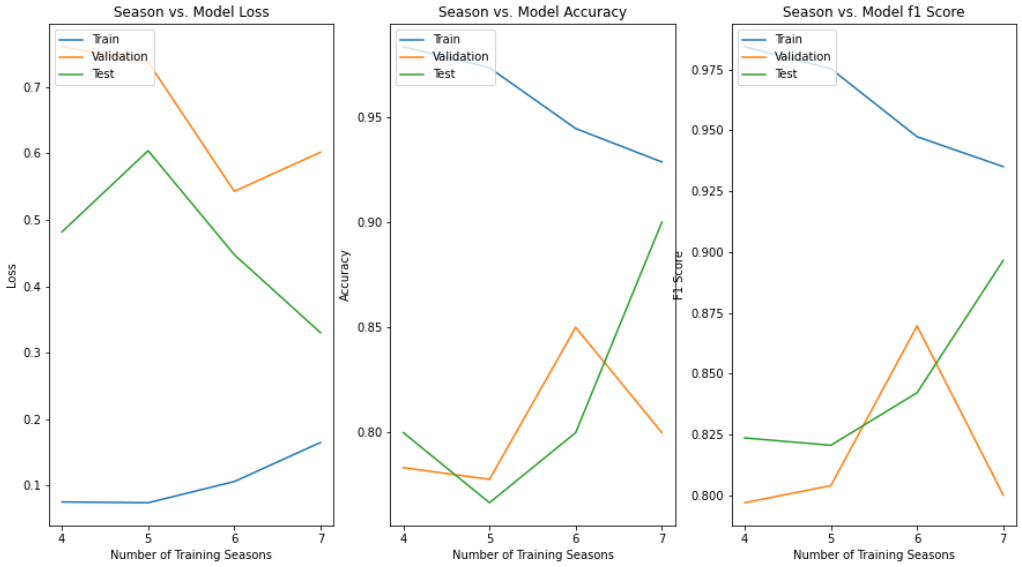
GraphSAGE outperformed the traditional GCN by around 20%. The best aggregator for our model was the mean.

rawSageMean - Test Accuracy: 0.8

Team	Probability	Actual Rank	Predicted Rank
SAS	0.9944	13	1
MIL	0.9936	1	2
UTA	0.9924	8	3
POR	0.9892	6	4
GSW	0.9885	3	5
BOS	0.9881	9	6
HOU	0.9869	5	7
TOR	0.9865	2	8
PHI	0.9826	7	9
LAC	0.9776	12	10
OKC	0.9662	10	11
ORL	0.9639	15	12
DET	0.9595	16	13
DEN	0.9557	4	14
IND	0.9362	11	15
NOP	0.8968	24	16
BRK	0.88	14	17
SAC	0.8684	19	18
MIN	0.8614	21	19
WAS	0.8499	25	20
LAL	0.8355	20	21
CHO	0.7757	17	22
DAL	0.4802	22	23
ATL	0.3135	26	24
MEM	0.1642	23	25
PHO	0.1468	29	26
MIA	0.0477	18	27
CLE	0.044	28	28
CHI	0.0101	27	29
NYK	0.0086	30	30

It was able to predict 80% of the rankings correctly, but we believe that the model has much room for improvement. We were unable to test the MaxPooling and MeanPooling aggregators as we'd like because of the time-constraint. Both of those models took a long time to run and we were unable to use them to their full potential. Despite this, we are satisfied with the accuracy we found.

Training Seasons Performance



We believe that the number of training and validation seasons used is extremely important. The difficulty in doing this is that validation sets are required for training sets, but ideally we would also like to use the validation sets as training for our test sets. This is because the most recent years from the test set would have the biggest influence in making predictions. We were unable to test this due to our time constraint, but we were able to determine that with our data of 10 seasons, using all of the data would lead to our best accuracies. This means that in the future, maybe if we include more than 10 seasons then the model accuracy would increase even more. This should be easy to implement as we already have the base code to retrieve and preprocess that data. We are also confident that our model will be able to scale with this increase in data perfectly if using the mean aggregator.

Conclusion & Discussion:

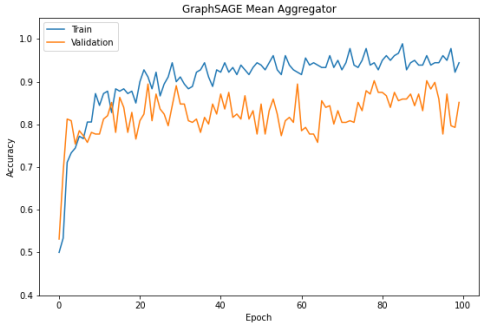
We were able to develop a model that inputs player stats, team rosters, and team schedules that would predict the rankings of each team. When using data from the 2011-2019, we were able to predict the ranking of the 2019 teams with 80% accuracy with GraphSAGE's mean aggregator. This model took around an hour to run, but MaxPooling and MeanPooling had to run overnight, which led us to not be able to experiment with them as much. This means that we have not been able to fully explore that effectiveness of other aggregators which may prove to be better than our current model. An improvement that we feel would be most effective is to incorporate validation sets into the training sets for test sets because recents years of the test set should influence the rankings more. There is also much room for feature selection as we had a very simple feature selection/aggregation. The importance of star players means a lot on a team, an our data pipeline only records that stats of the best player stats, but we believe that it would be important to record the stats of the 5 best players, the starters because these are the players that will get the most minutes in the game, thus the most influence on the game/team. The model could also be modified to look at more specific predictions rather than just seedings. We feel that it would be more useful to predict the outcome of matches as this is information that would be able to produce results and be used almost immediately. We are confident that with more research using

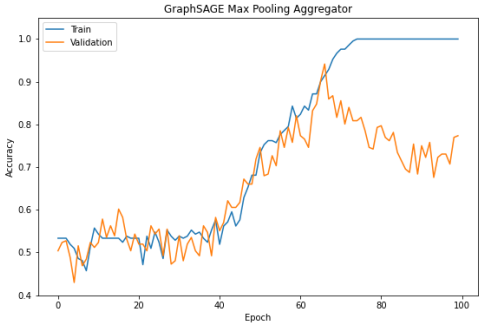
Graph Neural Networks will drastically improve accuracy in predicting rankings as well as have an impressive accuracy in predicting many other labels.

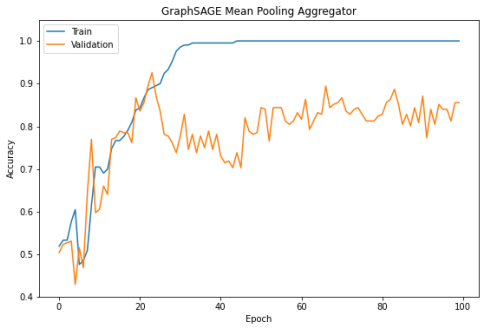
References:

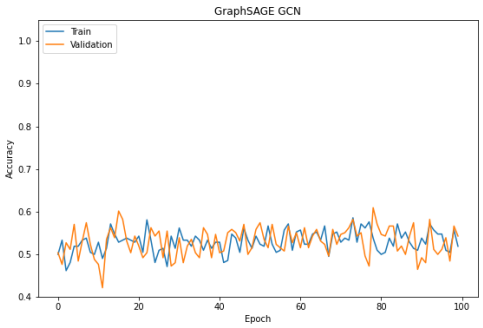
Inductive Representation Learning on Large Graphs. W.L. Hamilton, R. Ying, and J. Leskovec *arXiv*:1706.02216 *[cs.SI]*, 2017.

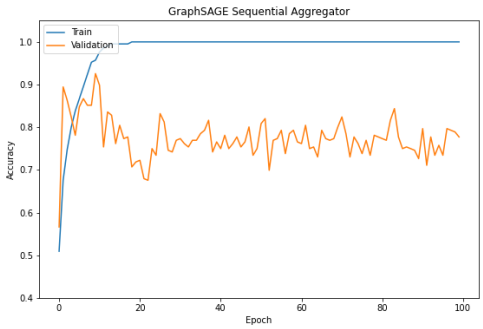
Appendix:

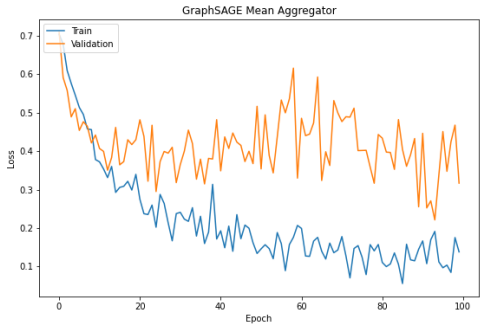


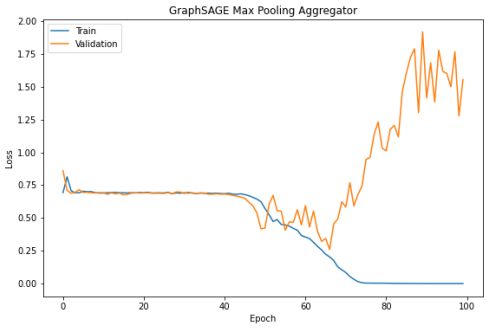


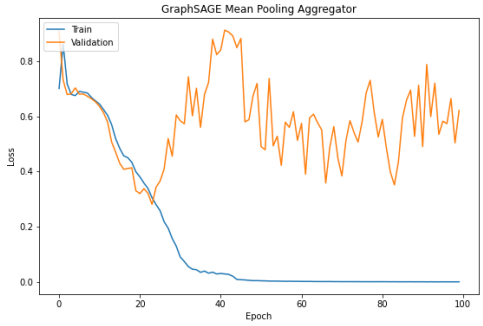


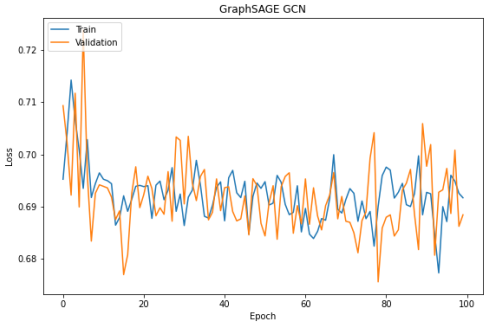


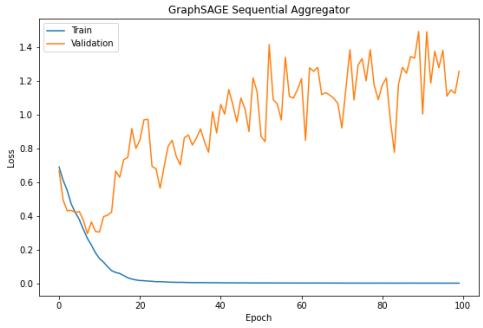



GraphSAGE Mean Aggregator
Train
Validation
Loss
Epoch


GraphSAGE Max Pooling Aggregator
Train
Validation
Loss
Epoch


GraphSAGE Mean Pooling Aggregator
Train
Validation
Loss
Epoch


GraphSAGE GCN
Train
Validation
Loss
Epoch


GraphSAGE Sequential Aggregator
Train
Validation
Loss
Epoch

rawGCN - Test Accuracy: 0.5333

Team	Probability	Actual Rank	Predicted Rank
CHI	0.5755	27	1
NYK	0.5745	30	2
BOS	0.5733	9	3
MIL	0.5729	1	4
WAS	0.5718	25	5
SAS	0.5714	13	6
MEM	0.5692	23	7
IND	0.569	11	8
SAC	0.5687	19	9
CHO	0.5686	17	10
BRK	0.5681	14	11
LAL	0.5674	20	12
POR	0.5674	6	13
PHI	0.5672	7	14
MIA	0.567	18	15
DEN	0.5668	4	16
ATL	0.5661	26	17
NOP	0.5661	24	18
GSW	0.5654	3	19
ORL	0.5648	15	20
MIN	0.5646	21	21
UTA	0.5637	8	22
CLE	0.5634	28	23
DAL	0.5629	22	24
OKC	0.5626	10	25
TOR	0.5621	2	26
LAC	0.5609	12	27
HOU	0.5606	5	28
DET	0.5578	16	29
PHO	0.5546	29	30

rawSageMeanPool - Test Accuracy: 0.7

Team	Probability	Actual Rank	Predicted Rank
SAS	0.9993	13	1
IND	0.9993	11	2
UTA	0.9993	8	3
MIL	0.9993	1	4
DEN	0.9993	4	5
MIN	0.9993	21	6
TOR	0.9993	2	7
OKC	0.9992	10	8
POR	0.9992	6	9
ORL	0.9992	15	10
PHI	0.9992	7	11
BOS	0.9992	9	12
NOP	0.9992	24	13
LAC	0.9992	12	14
GSW	0.9992	3	15
DET	0.9991	16	16
LAL	0.9991	20	17
HOU	0.9991	5	18
SAC	0.9988	19	19
CHO	0.9967	17	20
WAS	0.9954	25	21
MEM	0.8893	23	22
DAL	0.763	22	23
BRK	0.2185	14	24
MIA	0.0039	18	25
ATL	0.0008	26	26
CLE	0.0008	28	27
PHO	0.0008	29	28
CHI	0.0007	27	29
NYK	0.0007	30	30

rawSeq - Test Accuracy: 0.7333

Team	Probability	Actual Rank	Predicted Rank
MIL	0.9991	1	1
SAS	0.9991	13	2
TOR	0.9991	2	3
UTA	0.9991	8	4
IND	0.9991	11	5
MIN	0.9991	21	6
OKC	0.9991	10	7
POR	0.9991	6	8
PHI	0.9991	7	9
GSW	0.9991	3	10
DEN	0.9991	4	11
LAC	0.9991	12	12
BOS	0.9991	9	13
ORL	0.9991	15	14
HOU	0.9991	5	15
NOP	0.9991	24	16
LAL	0.999	20	17
DET	0.9989	16	18
SAC	0.9988	19	19
WAS	0.998	25	20
CHO	0.9547	17	21
MEM	0.7883	23	22
BRK	0.023	14	23
DAL	0.0024	22	24
MIA	0.0013	18	25
ATL	0.0012	26	26
PHO	0.0012	29	27
CLE	0.0012	28	28
CHI	0.001	27	29
NYK	0.001	30	30

rawSageMaxPool - Test Accuracy: 0.7667

Team	Probability	Actual Rank	Predicted Rank
GSW	0.9996	3	1
MIL	0.9996	1	2
LAC	0.9996	12	3
POR	0.9996	6	4
OKC	0.9996	10	5
UTA	0.9996	8	6
IND	0.9996	11	7
SAS	0.9996	13	8
BOS	0.9996	9	9
TOR	0.9996	2	10
ORL	0.9996	15	11
DEN	0.9996	4	12
PHI	0.9996	7	13
MIN	0.9996	21	14
LAL	0.9996	20	15
NOP	0.9996	24	16
WAS	0.9995	25	17
CHO	0.9995	17	18
SAC	0.9992	19	19
HOU	0.9992	5	20
DET	0.999	16	21
MEM	0.9929	23	22
BRK	0.9111	14	23
MIA	0.0061	18	24
DAL	0.0033	22	25
ATL	0.0006	26	26
CLE	0.0006	28	27
PHO	0.0006	29	28
CHI	0.0006	27	29
NYK	0.0005	30	30

rawSageMean - Test Accuracy: 0.8

Team	Probability	Actual Rank	Predicted Rank
SAS	0.9944	13	1
MIL	0.9936	1	2
UTA	0.9924	8	3
POR	0.9892	6	4
GSW	0.9885	3	5
BOS	0.9881	9	6
HOU	0.9869	5	7
TOR	0.9865	2	8
PHI	0.9826	7	9
LAC	0.9776	12	10
OKC	0.9662	10	11
ORL	0.9639	15	12
DET	0.9595	16	13
DEN	0.9557	4	14
IND	0.9362	11	15
NOP	0.8968	24	16
BRK	0.88	14	17
SAC	0.8684	19	18
MIN	0.8614	21	19
WAS	0.8499	25	20
LAL	0.8355	20	21
CHO	0.7757	17	22
DAL	0.4802	22	23
ATL	0.3135	26	24
MEM	0.1642	23	25
PHO	0.1468	29	26
MIA	0.0477	18	27
CLE	0.044	28	28
CHI	0.0101	27	29
NYK	0.0086	30	30