DATA SCIENCE

Predicting the outcome of games with Machine Learn

How we used (and you can too) machine learning to l the role statistics play in sports.

Josh Weiner
Jan 5, 2021 19 min read

When deciding on a final project for our Big Data my partners Jack Rosener, Jackson Joffe and I l an interest in sports with the principles learned semester. After a few days of discussion, we set that would aim to predict the outcome of NBA g implementing our goal, we found it helpful to di into the following steps with the following ques

1. **Scraping Relevant Data** – where do we gath team and player statistics over several seas
2. **Cleaning and Processing the Data** – how ca combine our scraped data so that it is both usable?

3. **Feature Engineering** – what additional metrics can we append to our datasets that would help any user or ML model to better understand and predict, respectively, relations within the data that may better inform our predictions?

5. **Predictions** – which models and features w useful for us in developing an accurate pred focus on team or aggregated individuals' sta

LATEST
EDITOR'S PICKS
DEEP DIVES
NEWSLETTER
WRITE FOR TDS
Sign in
Submit an Article

Before we get into the nitty-gritty of our workflo moment to review and acknowledge the other w exact topic. First, in 2013 Renato Torres from the Wisconsin-Madison set out to accomplish a sim did and predict specific season outcomes of NB different machine learning models. He used sev featured in our project, primarily feature reducti multicollinearity from the available data, and als different models to explore those with the highe Like our project, his selected features included but unlike our project had a particular focus on percentage at home and away. (We will explore analysis later.) A lot of other fantastic work on t before and can be read here:

Cheng, Ge & Zhang, Zhenyu & Kyebambe, Mos Kimbugwe. (2016). Predicting the Outcome of Based on the Maximum Entropy Principle. Jor Predicting the Outcomes of NBA Games. Nort

University. Fayad, Alexander. Building My First Machine Learning Model | NBA Prediction Algorithm. Towards Data Science. The Complete History of the NBA. FiveThirtyEight.

Through our research, we found that the best published model had a prediction accuracy of 74.1% (for playoff outcomes), with most others achieving an upper bound between 66–72% accuracy. Most published research, too, focused on predicting playoff scores – which may lend towards biased teams are more consistent in a number of stats regular season, and playoff game expected outc experience less variance as a result. Critically, n upset rate across the entire season in the NBA a the playoffs, the upset rate – as defined by tear regular season record winning— drops to 22% (v means most NBA-playoff prediction models *una* Because our project looked to predict the outco game and is playoff agnostic, we were looking to that could reach and hopefully beat a 67.9% acc doing so predict some upsets.

Feel free to follow along here or view our files o

Scraping our Data

We scraped our data from available information — which has extremely detailed team and playe game played since the 2008–2009 season. Scra days due to rate-limiting (as we had to query th

game over 12 seasons), and was initially compiled into a JSON format and finally saved to csv files.

> **Author's Note:** We accessed data from Synergy Sports with funding from the University of Pennsylvania. As an unfortunate consequence of this, we are not at liberty to provide public access to our dataset. However, we have compiled a list of alternative datasets through which one can replicate and improve upon our data: https://www.basketba
> reference.com/leagues
> https://www.kaggle.com/datasets/wyattowalsl
> https://www.kaggle.com/datasets/nathanlauga
> https://www.kaggle.com/datasets/drgilermo/n

Cleaning the Data

We now had both player stats and team stats fo
season saved as seperate csv files. Our next ste
all this data and combine it into two large dataf
containing the player stats for the past 12 seaso
containing the team stats. Once created, we wo
dataframes to remove invalid statistics (negative
columns that served little purpose to us (charge
taken/committed, for example).

We then saved these new dataframes to the foll
which allowed us (and you) to skip the laborious
steps of both scraping and cleaning the data wh
notebook's runtime:

Search this file…

	Date	Game_ID	Season	H_Team	A_Team	H_Team_Record
0	2020-08-14	732811	2019 - 2020	Houston Rockets	Philadelphia 76ers	44 - 27
1	2020-08-14	732814	2019 - 2020	Toronto Raptors	Denver Nuggets	52 - 19
2	2020-08-14	732812	2019 - 2020	Indiana Pacers	Miami Heat	44 - 28
3	2020-08-13	732808	2019 - 2020	Orlando Magic	New Orleans Pelicans	32 - 40
4	2020-08-13	732809	2019 - 2020	Phoenix Suns	Dallas Mavericks	33 - 39
5	2020-08-13	732806	2019 - 2020	Los Angeles Lakers	Sacramento Kings	52 - 18
6	2020-08-13	732805	2019 - 2020	Brooklyn Nets	Portland Trail Blazers	35 - 36
7	2020-08-13	732807	2019 - 2020	Memphis Grizzlies	Milwaukee Bucks	33 - 39
8	2020-08-13	732810	2019 - 2020	Utah Jazz	San Antonio Spurs	43 - 28
9	2020-08-13	732804	2019 - 2020	Boston Celtics	Washingto	

NBA_Combined_Team_Stats.csv hosted with ❤ by GitHub

Search this file…

	PlayerName	PlayerTeam	OpposingTeam	Dat
0	PJ Tucker	Houston Rockets	Philadelphia 76ers	202
1	Hamidou Diallo	Oklahoma City Thunder	Los Angeles Clippers	202
2	Danilo Galinari	Oklahoma City Thunder	Los Angeles Clippers	202
3	Nerlens Noel	Oklahoma City Thunder	Los Angeles Clippers	202
4	Steven Adams	Oklahoma City Thunder	Los Angeles Clippers	202
5	Devon Hall	Oklahoma City Thunder	Los Angeles Clippers	202
6	Kevin Hervey	Oklahoma City Thunder	Los Angeles Clippers	202
7	Dennis Schroder	Oklahoma City Thunder	Los Angeles Clippers	202
8	Andre Roberson	Oklahoma City Thunder	Los Angeles Clippers	202
9	Terrance Ferguson	Oklahoma City Thunder	Los Angeles Clippers	202

NBA_Combined_Player_Stats.csv hosted with ❤ by GitHub

Feature Engineering

This is where the fun stuff begins. Our primary g
all of the available data understandable: game-b
for an entire team don't help us much unless w
data in a higher-level analysis that leads us to o
predicting wins and losses. To that end, we soug
different features which we would use in unders
teams progressed and regressed throughout eac

1. **Elo Ratings**

This is perhaps the best existing method to relativize NBA team strength and performance over many seasons. The way Elo Ratings are calculated is simple: all teams start at a median score of 1500 and are either given or subtracted points based on the final score of each game and where it was played with weights being given to point difference, upsets, and location. In essence, it's a more sophisticated win-loss reco prediction models don't look at Elo Ratings but amalgamate a simple win-loss record with seve We wanted to use Elo to appropriately weight q losses), while also recognizing that not all team equal. The exact formula is as follows: If R_i is t rating of a team, its Elo rating after its played it defined as follows:

$$R_{i+1} = k * (S_{team} - E_{team} + R_i)$$

We calculate ELO for each team and each game for every season of

Here, _*Steam* is a state variable: 1 if the team w loses. E_team represents the expected win prob team, which is represented as:

$$E_{team} = \frac{1}{1 + 10^{\frac{opp\_elo - team\_elo}{400}}}$$

k is a moving constant, dependent on both the and difference in Elo ratings:

$$k = 20\frac{(MOV_{winner} + 3)^{0.8}}{7.5 + 0.006(elo\_difference_{winner})}$$

It's also important to note that Elo ratings carry over from season-to-season (as all teams are not created equal, good teams tend to stay good or at least gradually decline – very rarely do teams drop onto or off of the map). If R represents the final Elo of a team in one season, it's Elo Rating at the beginning of the next season is approximately:

$$(R * 0.75) + (0.25 * 1505)$$

We can actually take a look at this metric over t
selecting three teams to view, and immediately
get key insights about the strength of teams thr

Here we can actually see that Elo Ratings track quite well with teams' perforn
years in which Golden State and Cleveland appeared and dueled in the NBA Fi
peaks of their Elo Ratings. We can also see what was widely confirmed by mo
time: that the Western Conference was much tougher than the East – as exhibi
wins on Elo for the Warriors versus the Cavaliers. We can also see how far thes
championship seasons and as they all suffered from roster losses and inj

2. **Recent Team Performance (Avg. stats over 10 most recent games)**

These are pretty self-explanatory, we are simply looking to average the stats for each team over their last 10 games. To do this we wrote a simple function that would calculate a sliding average for a given team's stats and a window of n games:

```
#given a team and a date, this method will return that team

def get_avg_stats_last_n_games(team, game_date, season_team
  prev_game_df = season_team_stats[season_team_stats['Date'

  h_df = prev_game_df.iloc[:, range(3, 43, 2)]
  h_df.columns = [x[2:] for x in h_df.columns]
  a_df = prev_game_df.iloc[:, range(4, 44, 2)]
  a_df.columns = [x[2:] for x in a_df.columns]

  df = pd.concat([h_df, a_df])
  df = df[df['Team'] == team]
  df.drop(columns = ['Team'], inplace=True)

  return df.mean()

recent_performance_df = pd.DataFrame()

for season in team_stats['Season'].unique() :
  l = ['Date', 'Game_ID', 'Season', 'H_Team', 'A_Team']
  other = list(team_stats.columns[9:47])
  cols = l + other

  season_team_stats = team_stats[team_stats['Season'] == se

  season_recent_performance_df = pd.DataFrame()

  for index, row in season_team_stats.iterrows() :
    game_id = row['Game_ID']
    game_date = row['Date']
    h_team = row['H_Team']
    a_team = row['A_Team']

```

```
    h_team_recent_performance = get_avg_stats_last_n_games(h_team, game_date, seasor
    h_team_recent_performance.index = ['H_Last_10_Avg_' + x for x in h_team_recent_p

    a_team_recent_performance = get_avg_stats_last_n_games(a_team, game_date, seasor
    a_team_recent_performance.index = ['A_Last_10_Avg_' + x for x in a_team_recent_p

    new_row = pd.concat([h_team_recent_performance, a_team_recent_performance], sort
    new_row['Game_ID'] = game_id

    season_recent_performance_df = season_recent_performance_df.append(new_row, ignc
    season_recent_performance_df = season_recent_performance_df[new_row.index]


recent_performance_df = pd.concat([recent_performance_df,
```

get_stats.py hosted with ❤ by GitHub

After saving this data into a new dataframe, we separate each game (which contains stats for b away teams) into its own rows by team, which a group-by and aggregate team stats much more simplifies existing features. Finally, we added a column to include the most critical measureme wins and losses.

3. **Recent Player Performance (Avg. stats over games)**

We create our player_recent_performance dataf similar methods to the above section, this time players as opposed to teams. This created a dat player's performance over the past 10 games.

4. **Player Season Performance**

We also sought to include the average player st season: unlike teams, players themselves get in

and out of the rotation and it's perhaps more critical for us to understand how player performances in individual games track with their averages. We will use this later in our models to see if it will allow for accurate predictions on the team-level.

5. **Player Efficiency Ratings (PER)**

Critically, just as we had done with teams via Elo Ratings, we wanted to be able to relativize player performar that combines seemingly unrelated statistics. O we could use Hollinger's Player Efficiency Rating predict team performance by the aggregated PE players. In the NBA, it is easy for players to expe inflated or deflated statistics (such as points pe by virtue of the amount of playing time they get, players or versus starters, number of games play outlier performances. We did not want to rely so averages simply because of their ability to skew. problem by weighting certain in-game statistics number of minutes played, which creates a met player performance relative to the number of m Thus for each player, we added a column for PE according to the following formula:

PER = (FGM x 85.910 + Steals x 53.897 + 3PTM
46.845 + Blocks x 39.190 + Offensive_Reb x 39
34.677 + Defensive_Reb x 14.707 – Foul x 17.174
20.091 – FG_Miss x 39.190 – TO x 53.897) x (1 /

Data Analysis

Our data analysis was centered around the use of Elo Ratings as our test metric. Essentially, could we be confident that Elo correlates with and correctly aggregates other statistics? Furthermore, would it be more appropriate for us to predict game outcomes using team stats (Elo Ratings) or averaged player stats (PER Ratings)?

First, let's explore the density of Elo Ratings acr[illegible] the NBA on a per-season basis. This tells us a little about the [illegible] across the league: if we can see Elo Ratings app[illegible] distribution that would suggest the league's tea[illegible] well-matched. Otherwise, we see large disparit[illegible] development of "super-teams".

LATEST

EDITOR'S PICKS

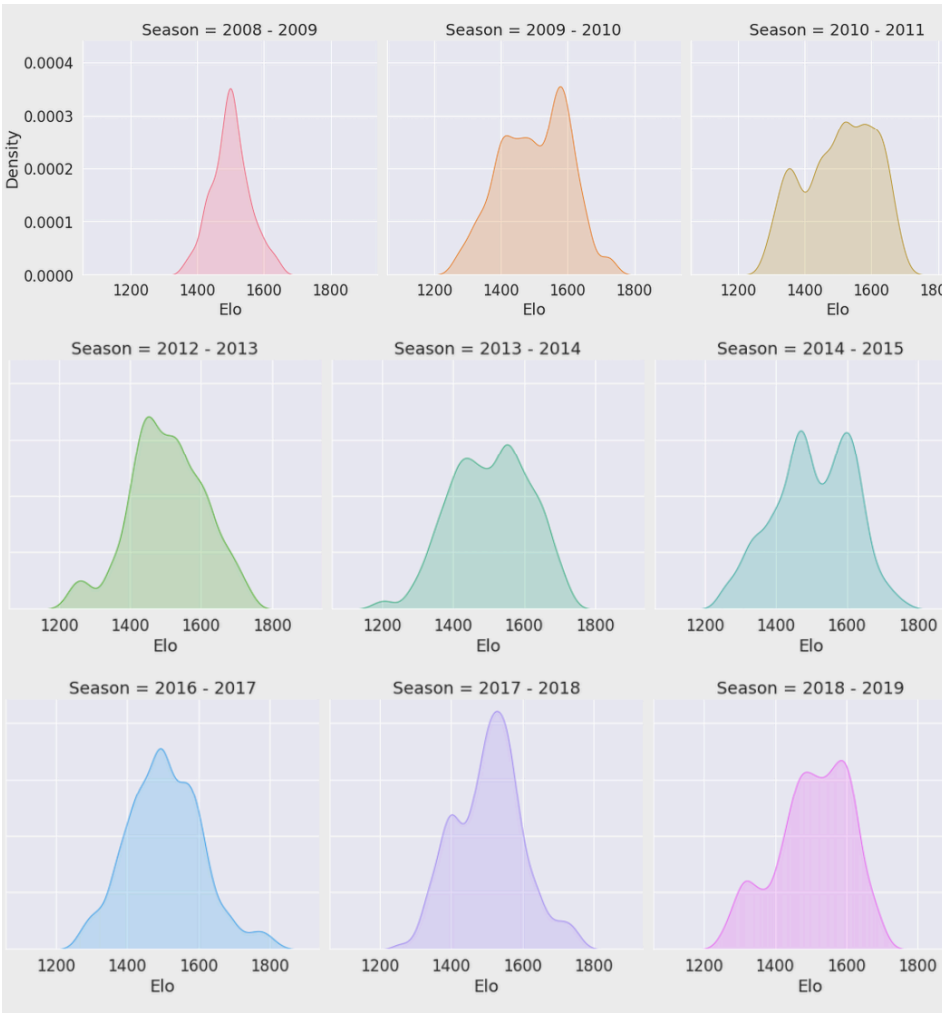
DEEP DIVES

NEWSLETTER

WRITE FOR TDS

Sign in

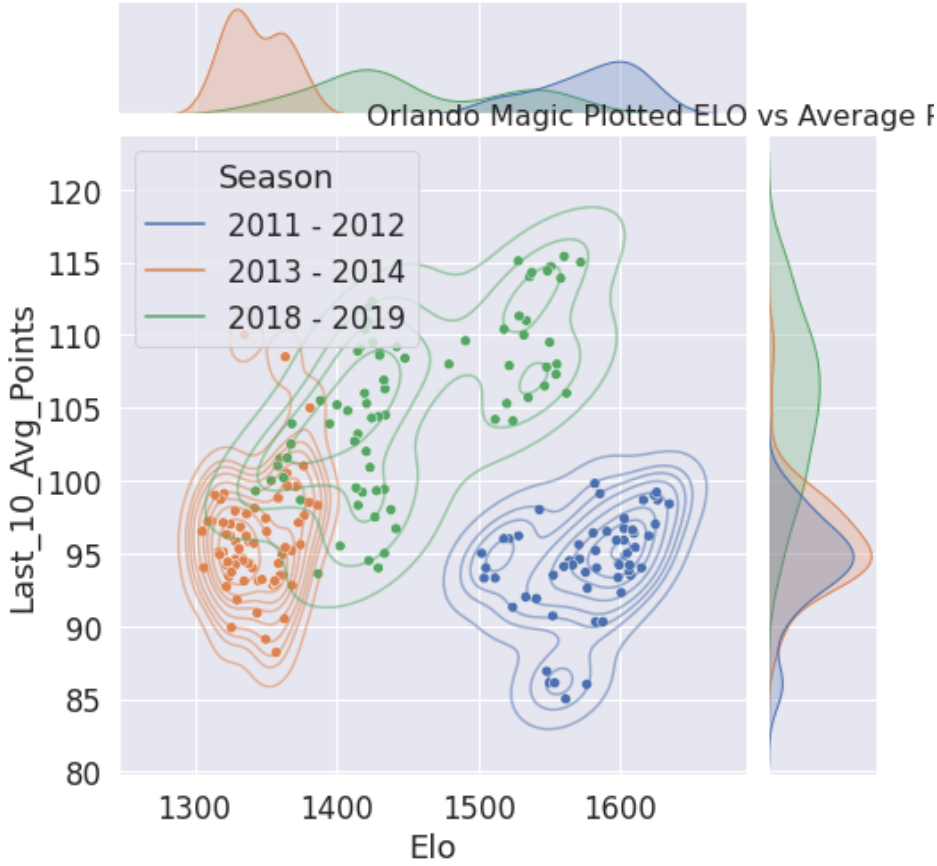
Submit an Article



Twelve seasons of league Elo densities. (Image by Aut

Moving away from an understanding of Elo Ratings from a league-perspective, we endeavored to see how Elo Ratings tracked against an individual team's performance in other statistics.

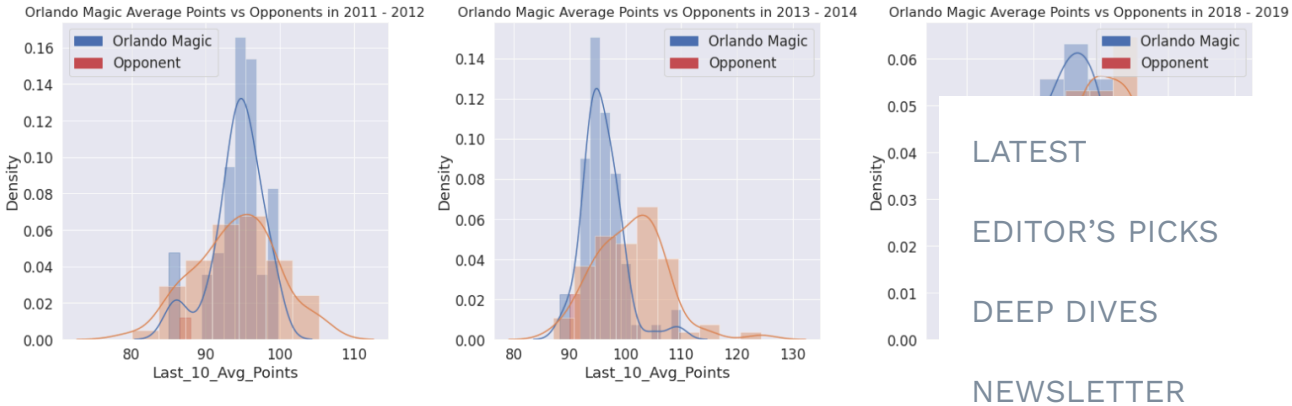
First, we looked to plot the distribution of Elo for a random team against the average number of points scored in recent games:



(Image by Author)

We can actually see from this that there is some
between the average number of points a team s
Elo Rating – the higher the average points score
window of games the higher the Elo Rating seen
However, we can also see that the Elo may also
variance across similar scoring figures. So, to be
how Elo Ratings track with points scored, we ex
average points scored compares to season aver

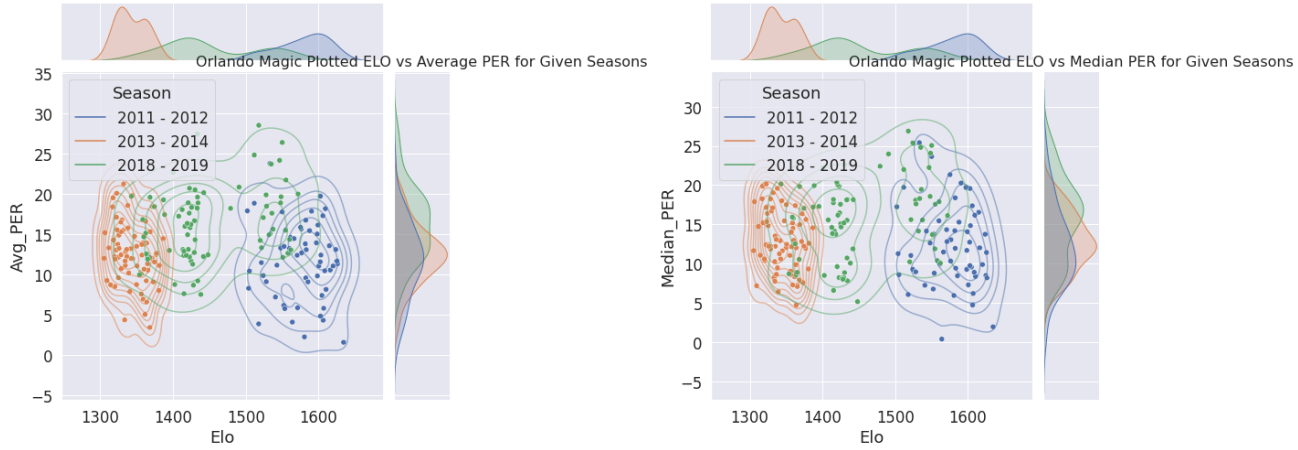
league – from there we can determine if points scored improves ELO, provided that high scoring is relative to the rest of the league. To do this, let's look at that same team for the same seasons and plot the distribution of points scored against its opponents.



(Image by Author)

This confirms our suspicions, as we can see tha distribution of average points is greater than tho opponents, or is more concentrated at an equal the Elo is higher for those seasons. When the gr an even or lesser value, those seasons' Elo ratin team are lower. Therefore, average points score solid determinant of predicting game outcomes, ***relativized***. This demonstrated for us that Elo w better determinant in predicting wins for us tha by design a relative statistic.

Shifting away from team statistics, we sought to Elo tracks better with player performance than performance. To do this, we took a similar appro plotted Elo Ratings with average points scored f random team, this time with PER.



(Image by Author)

LATEST

EDITOR'S PICKS

DEEP DIVES

NEWSLETTER

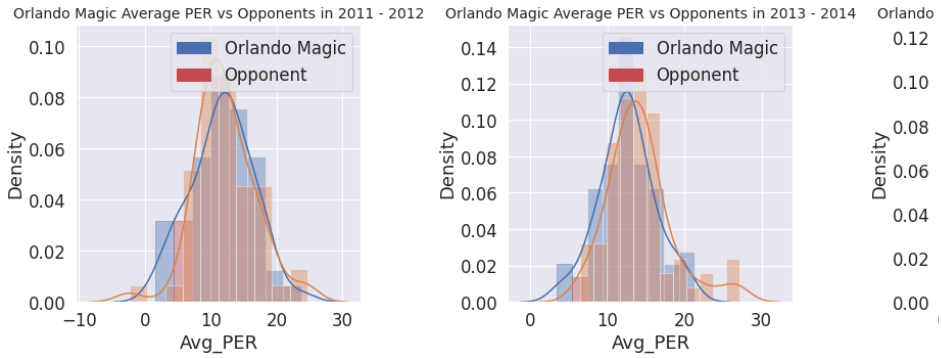
WRITE FOR TDS

Sign in

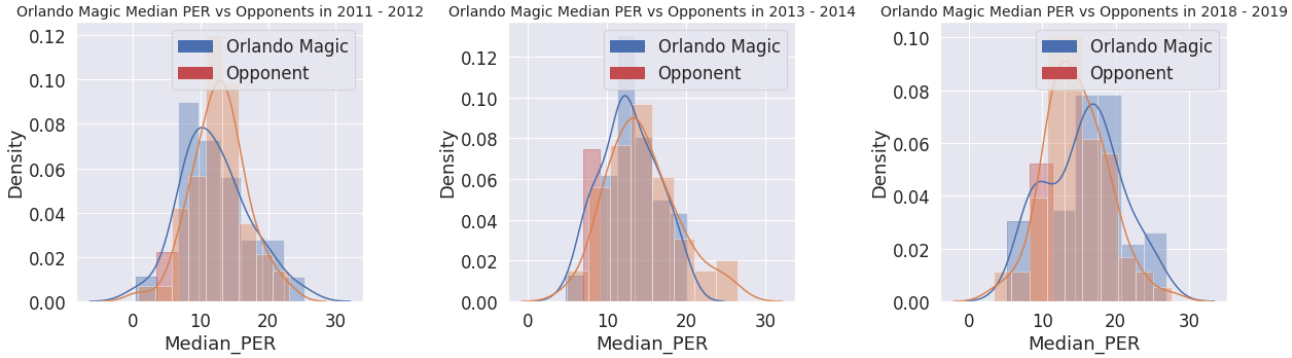
Submit an Article

From the plotted data, we can see that aggregat
compared to opponents doesn't show much of a
with the strength of a team as determined by El
points scored translates better – which makes s
player's efficiency isn't necessarily tied to scorir
points – and points scored against opponents is
of winning a game and therefore impacting Elo.

We can see this further by mapping the Orlando
and median PER ratings against its opponents fo
seasons, and find that there is almost no relatic
PER averages or medians and team strength.



From these and the above plotted distribution, we see that Average PER – while Elo Ratings throughout the season – generally show us little about how individ team strength when tracked against opponents. (Image by



Median PER ratings show even less of a correlation to Elo Ratings throughout a given season. Here we can observe that in winning seasons (2011–12), the Orlando Magic had a lower median PER than its opponents for most games, yet they had their highest Elo Ratings and best record in recent [illegible] (Image by Author)

From all of our analysis of relativized team versu
player statistics, it looks clear to us that our Elo
determinants would be better features to train
when it comes to predicting the outcome of NB

Predicting the Outcome of Games B
Team Statistics and Elo Ratings

Our first step here was to split our data into fea
columns. Reading from our dataset, once split,
sklearn to randomly split our data into train and
80:20 ratio.

```
from sklearn.model_selection import train_test_split

final_team_stats = pd.read_csv('/content/drive/MyDrive/CIS5

cols = [0,1,2,3,4,7,8, 18, 37]
final_team_stats.drop(final_team_stats.columns[cols],axis=1

features = final_team_stats.drop(columns = 'Label')
label = final_team_stats['Label']

x_train, x_test, y_train, y_test = train_test_split(feature
```

model_prep.py hosted with ❤ by GitHub

The first model we aimed to use to predict the outcome of an NBA game was a Logistic Regression model. Unlike a Linear Regression model which predicts outcomes on a range of values between (and sometimes outside) 0 and 1, Logistic Regression models aim to group predictions into binary outcomes. Since we are predicting wins and losses, this type of classification suits us perfectly.

To begin, we used a simple non-parameterized l
team stats and Elo Ratings as parameters using

After playing around with some hyperparameter
that using max_iter=131 and verbose=2 slightly ir
initial testing accuracy to 66.95%. Definitely not
parameterized model and very close to our desi
accuracy. However, we sought to see if we coulc
hyperparameters to improve our overall accurac
would try out many combinations of possible hy
our data to give us the absolute best weights fo


```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

# create a simple, non-parameterized Logistic Regression mo
model = LogisticRegression(random_state=42)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
print(metrics.accuracy_score(y_test, y_pred))

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(model.get_params())

# create complex Logistic Regression with max_iter=131
```

```
log_model = LogisticRegression(max_iter=131, verbose=2, random_state=42)
log_model.fit(x_train, y_train)
y_pred_log = log_model.predict(x_test)
print(metrics.accuracy_score(y_test, y_pred_log))
```

team_logistic_regression.py hosted with ❤ by GitHub view raw

We accomplished this using cross-validation: because we only have a vague idea of the parameters we might want to use, our best approach is to narrow our search is and evaluate a wide range of values for each hyperparameter.

Using RandomizedSearchCV, we searched amon
= **59,400** possible settings – and so the most ef
this would be to take a random sample of the v

Running our model with the best parameter valu
samples actually decreased the accuracy of our
which showed us that while random sampling h
down our hyper parameter tuning within a distri
have to explicitly check all combinations with G


```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of grid se
# Penalty type
penalty = ['l1', 'l2', 'elasticnet', 'none']
# Solver type
solver = ['lbfgs', 'liblinear']
# Maximum number of iterations
max_iter = [int(x) for x in np.linspace(start = 80, stop =
# Multi class
multi_class = ['auto', 'ovr']
# Verbosity
verbose = [0, 1, 2]
# l1 ratio
l1_ratio = [0, 0.8, 0.9, 1]
# C
C = [0.5, 0.75, 1.0, 1.25, 1.5]
```

```
# Create the param grid
param_grid = {'penalty': penalty, 'solver': solver, 'max_iter':max_iter,
    'multi_class':multi_class, 'verbose':verbose, 'l1_ratio':l1_ratio,
    'C':C
}
pprint(param_grid)

# Instantiate the grid search model with 2-fold cross-validation
log_grid_search = GridSearchCV(estimator = LogisticRegression(random_state=42), para

# Fit the grid search to the data
log_grid_search.fit(x_train, y_train)
best_log_grid = log_grid_search.best_estimator_
best_log_grid.fit(x_train, y_train)
y_pred_best_log = best_log_grid.predict(x_test)
print(metrics.accuracy_score(y_test, y_pred_best_log))
```

grid_search.py hosted with ❤ by GitHub

In this case, implementing GridSearch only marg
our accuracy with our LR model.

The second model we looked to implement was
RandomForestClassifier, which can be efficientl
regressions and classifications. In this case, we
Classifier can build a proper decision tree to de
the given team-stats.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(x_train, y_train)
y_pred_rf = rf.predict(x_test)
print(metrics.accuracy_score(y_test, y_pred_rf))

from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

team_random_forest.py hosted with ❤ by GitHub

Immediately, we get that the RandomForestClassifier reaches an initial accuracy of 66.95%, which again is pretty good. Like with the LR model, we attempted to tune the hyperparameters to give us more accurate results – first using RandomizedSearchCV.

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 0, stop
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(0, 100, num = 6)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rfc = RandomForestClassifier(random_state=42)
# Random search of parameters, using 2-fold cross validatio
# search across 100 different combinations, and use all ava
rfc_random = RandomizedSearchCV(estimator = rfc, param_dist
# Fit the random search model
rfc_random.fit(x_train, y_train)
y_pred_rfr_random = rfc_random.predict(x_test)
print(metrics.accuracy_score(y_test, y_pred_rfr_random))
```

randomized_search.py hosted with ❤ by GitHub

Unlike with the LR model, we find that RandomizedSearch improves our hyperparameter tuning, giving us a better accuracy of 67.15%.

Running GridSearchCV in a similar manner to what we did above, we also sought to explicitly test 2 *1* 6 *2* 3 *3* 5 = 1080 combinations of settings instead of randomly sampling a distribution of settings. GridSearch also gave us an improvement from the base RandomForestClassifier, with an accuracy of 67.1

Overall, when running both a LinearRegression a RandomForestClassifier on the team stats and E achieved a win-prediction accuracy of **66.95%–6** basketball games, which as we established earli variable in their actual versus predicted results, significant result.

Predicting the Outcome of Games B Individual Player Statistics and Sco

We then took a different approach to predicting game to see if we can achieve any better perfon larger dataset of individual player statistics that we will train a model to predict how many point score in a given game. We will predict this base average season stats up until the game we are t as well as their average performance over the p already created this data in the feature engineer above. We will also make use of Elo ratings in ou well, as presumably the higher rating of the opp less points a player will score. Once we have thi

predict how many points a team will score in a game by summing the predicted number of points of each individual player will score. With this information we will be able to predict which team will score more points and thus win the game.

Before we run our models, we need to clean our data slightly. For some games in this dataset, we have the statistics for one teams' players, but not for the other team – generally only for the first game that other team plays in the season. Thus, all these games from the dataset.

```
game_id_value_counts = final_player_stats['GameID'].value_co
valid_game_ids = [x for x in final_player_stats['GameID'] if

final_player_stats = final_player_stats[final_player_stats['
```

cleaning_player_stats.py hosted with ❤ by GitHub

Unlike with the above games, we can't randomly into train and test sets. We are looking to use ir statistics to predict the final score of a team, th all players playing in the same game together. To split up our train and test sets by game so playe same game stay together. About 80% of the gam train set and 20% will be in the test set:

```
import random
games = set(final_player_stats['GameID'].unique())

train_set_game_ids = random.sample(games, 9700)

x_train = features[features['GameID'].isin(train_set_game_i
y_train = label[label['GameID'].isin(train_set_game_ids)].d

x_test = features[features['GameID'].isin(train_set_game_id
y_test = label[label['GameID'].isin(train_set_game_ids) ==
```

train_test_split.py hosted with ❤ by GitHub view raw

Instead of using a Logistic Regression model, for player scoring we will use a Linear Regression model as we are looking to predict a range of possible values (points scored) instead of simply predicting a win or a loss. Our RMSE (Root Mean Squared Error) for all players was 5.56, or the equivalent of each player making or missing around 2–3 baskets game arc averages.

On the test set, we grouped each team's predict each game and compared it with their actual sc Computing the numbers of games won versus th on predicted scoring gave us a ratio of 1483/252 of **58.66%**. Clearly, and as we realized earlier wh distributions of teams versus their opponents, a performance is too variable of a determinant to predict the outcome of games – especially wher team performance which tends to be more cons games.

Conclusions and Future Considerati

As avid NBA fans, we felt that creating a model t outcome of NBA games would be an interesting taught us a lot about building classifiers for pro game outcomes. We were able to utilize many o learned in our Big Data Analytics class for this p scraping, data cleaning, feature analysis, buildin hyperparameter tuning – and want to thank Pro fantastic work in teaching throughout the seme

Our Random Forest Regression model, with parameters optimized through RandomSearchCV, gave us the highest testing accuracy of 67.15%. It is slightly higher than the Logistic Regression model, and it is much higher than the Linear Regression model based on individual player statistics. Optimizing parameters using GridSearchCV and RandomizedSearchCV was time consuming and computationally costly, and it resulted in only marginal changes in testing accuracy. If we had more tim
spend less time optimizing parameters and mor
model.

The best NBA game prediction models only accu
winner about 70% of the time, so our logistic re
and random forest classifier are both very close
bound of predictions that currently exist. If we h
would explore other models and see just how h
accuracy we could get. Some of those candidate
Classifier, linear discriminant analysis, convoluti
naïve Bayes classifier.

Hopefully, you enjoyed reading about our work a
enjoyed making it – and learned something fron

WRITTEN BY

Josh Weiner

See all from Josh Weiner

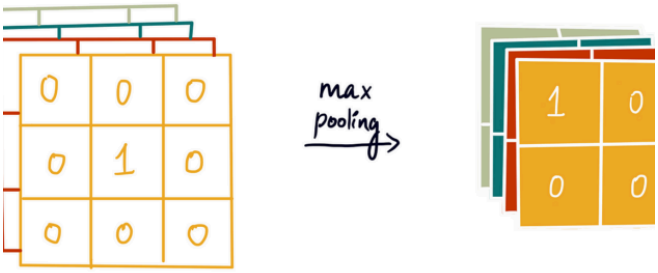
Data Science Machine Learning Python

Share This Article

Towards Data Science is a community publication. Submit your insights to reach our global audience and earn through the TDS Author Payment Progra

Write for TDS

Related Articles



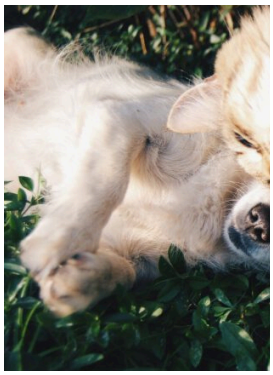
ARTIFICIAL INTELLIGENCE

Implementing Convolutional Neural Networks in TensorFlow

Step-by-step code guide to building a Convolutional Neural Network

Shreya Rao

August 20, 2024 6 min read



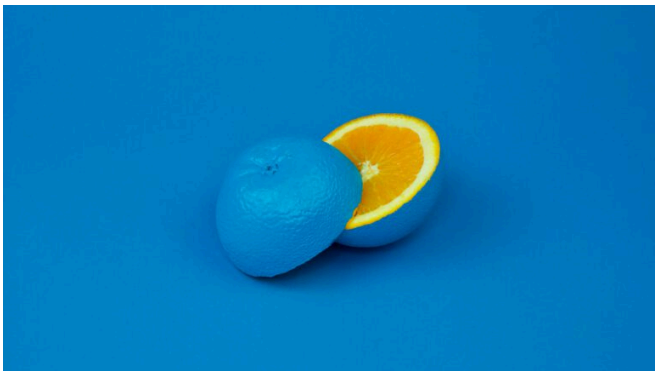
ARTIFICIAL INTELLI

How to Foreca Time Series

A beginner's guid reconciliation

Dr. Robert Kübler

August 20, 2024 1



DATA SCIENCE

Hands-on Time Series Anomaly Detection using Autoencoders, with Python

Here's how to use Autoencoders to detect signals with anomalies in a few lines of...

Piero Paialunga

August 21, 2024 12 min read

MACHINE LEARNING

3 AI Use Case (That Are Not a Chatbot)

Feature engineeri
unstructured dat

Shaw Talebi

August 21, 2024 7

DATA SCIENCE

Solving a Constrained Project Scheduling Problem with Quantum Annealing

Solving the resource constrained project scheduling problem (RCPSP) with D-Wave's hybrid constrained quadratic model (CQM)

Luis Fernando PÉREZ ARMAS, Ph.D.

August 20, 2024 29 min read

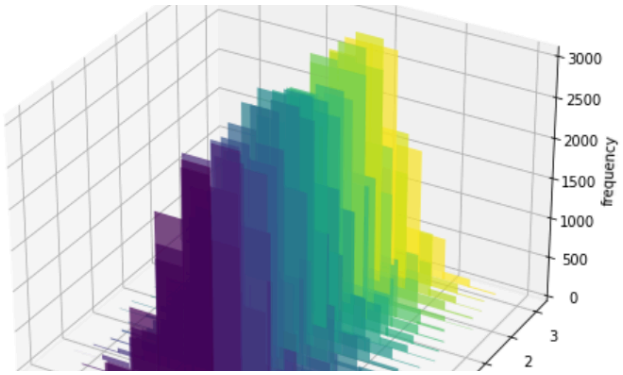
DATA SCIENCE

Back To Basic Regression ar

An illustrated gui
learning concept

Shreya Rao

February 3, 2023



DATA SCIENCE

Must-Know in Statistics: The Bivariate Normal Projection Explained

Derivation and practical examples of this powerful concept

Luigi Battistoni

August 14, 2024 7 min read