Victor Bornstein

Logistic Regression of 2018-2019 NBA All-Star Selections

Each year, halfway through the NBA season, a group of media, players, and fans vote for All-Stars-the players which they consider to be the best in the league. There is often debate as to what criteria or features go into an all-star selection, as many media members value team wins while casual fans only consider popularity or ppg.

I created a Logistic Regression model that predicts if an NBA player will be an All-Star based on the features of wins, ppg, rebounds, and assists. It is a binary model because of only two values of the response (1= all-star, 0= not all-star). Logistic Regression fits as a binary response is predicted by continuous/discrete independent variables. The goal is to find which features have the largest impact on all-star selection and create a model that accurately classifies the given data and can correctly classify unknown data.

Our data comes from https://github.com/cjporteo/ml-NBA-asg-predictor/blob/master/Data%20Collection/ASG_train.csv which contains 23 features of each individual player every season from 1996-2018, which can be found on stats.nba.com. In order to reduce the dimensionality of the data set I selected all 488 players from the 2018 NBA season and only considering the features Wins, ppg, assists, rebounds, and response all-star selection for the training data. This model is reasonable as I selected the most relevant features commonly considered in all stars. Note that because the model is based only from 2018 players it may not capture the true probability distribution for distant years as our features are time dependent (avg ppg increases over time), however it will likely still accurate in close years as they have minimal confounding variables (i.e. team changes) therefore we will compare the model with training data from 2018 to known all stars from 2019 to minimize error.

When the logistic regression model was applied, the largest coefficient was found to be Assists, implying that the features assists has the largest impact on being an all-star a possible reason for this is perhaps due to the prominence of guards in the all-star game who tend to pass more. All coefficients were found to be positive indicating they all contribute positively to an all-star appearance (i.e. more wins increase log odds of all-star selection). The model score was found to be 97% which implies 1 outlier of the ~40 all-stars. Ppg is commonly seen as the biggest factor in a player's success and when expanding ppg to a cubic polynomial, the beta coefficients of ppg^2 and ppg^3 were the smallest found implying minimal effect on our response.

Our p value in the ANOVA test was found to be ~0 which is proof that some features are relevant. To find the best model selection the AIC score was calculated for models with features of wins, ppg, wins and ppg, and no features. The lowest aic score of -1608 was found for our ppg model verifying our suspicion that ppg is correlated with all-star selection.

In order to test the accuracy of our model, the stats of LeBron James and Rudy Gobert, 2 all stars of the next year, were fed into our model and LeBron was found to have a pvalue of 99% whereas Rudy 38% probability of all-star. This is expected as LeBron is widely considered a top player in the league (top 5 in ppg and assists) almost guaranteeing an all-star appearance.

The low probability of Rudy all-star although he was an all-star is likely due to unexplained features in the model. He is likely to be voted in due to his high bpg and defensive stats which we removed in our feature selection leaving them unexplained. Therefore, we can conclude that our model is biased towards selecting offensive players due to our feature selection and would be better explained by adding features such as bpg, and spg.

```
import numpy as np
data = np.loadtxt('2018nba.csv', delimiter=',', usecols=(7,8,9,10,21))
```

```
data
```

```
array([[25. , 35.7,  6.4,  8.3,  1. ],
       [20. , 29.3, 13.3,  4.4,  1. ],
       [28. , 29.3,  5.2,  5.6,  1. ],
       ...,
       [ 1. ,  0. ,  1. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ,  0. ],
       [ 1. ,  0. ,  0. ,  0. ,  0. ]])
```

```
X = data[:,0:4]
Y = data[:,4]
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='newton-cg', penalty='none')
model.fit(X, Y)

model.coef_
```

```
array([[0.10958996, 0.32531141, 0.20496524, 0.40854394]])
```

```
model.intercept_
```

```
array([-12.98307602])
```

```
model.score(X, Y)
```

```
0.9774590163934426
```

```
X = np.hstack((np.ones((488,1)), data[:,0:4], data[:,1:2]**2,data[:,1:2]**3))
```

```
beta = np.linalg.inv(X.T@X)@X.T@Y
s2 = (Y-X@beta).T @ (Y-X@beta) / (488-7)
```

```
beta
```

```
array([ 4.13432206e-02,  2.69922844e-03, -4.20771539e-02,  1.25059134e-02,
        2.51360516e-02,  2.10862869e-03,  1.41920321e-06])
```

```
s2
```

```
0.024108031047212777
```

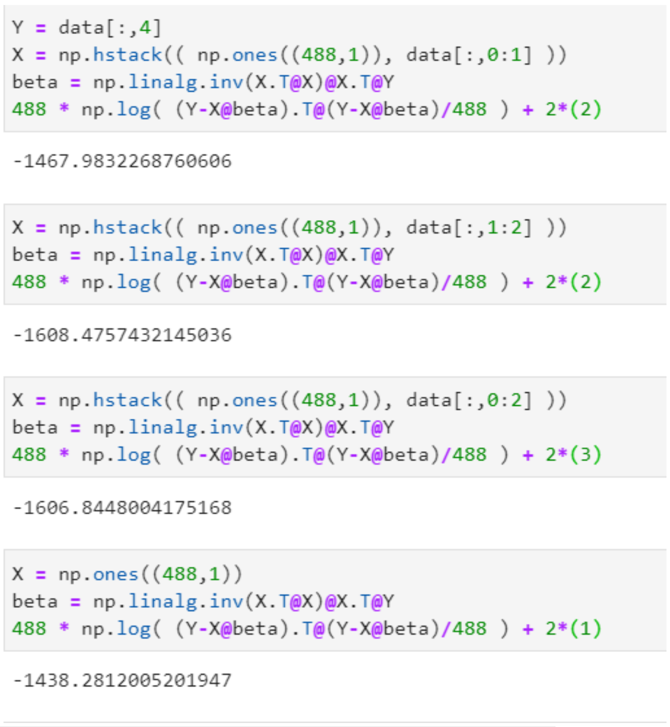
```
X = np.hstack((np.ones((488,1)),data[:,0:4]))
Y = data[:,4]
```

```
beta = np.linalg.inv(X.T@X)@X.T@Y
mu = X @ beta
e = Y - mu
s2 = e.T @ e / (488-5)
```

```
import scipy.stats as st
T = (mu.T@mu - 488*np.mean(Y)**2 ) /(5-1) /s2
p = 1-st.f.cdf(T, 5-1, 488-5)
```

```
p
```

```
1.1102230246251565e-16
```



```python
Y = data[:,4]
X = np.hstack(( np.ones((488,1)), data[:,0:1] ))
beta = np.linalg.inv(X.T@X)@X.T@Y
488 * np.log( (Y-X@beta).T@(Y-X@beta)/488 ) + 2*(2)
```

```
-1467.9832268760606
```

```python
X = np.hstack(( np.ones((488,1)), data[:,1:2] ))
beta = np.linalg.inv(X.T@X)@X.T@Y
488 * np.log( (Y-X@beta).T@(Y-X@beta)/488 ) + 2*(2)
```

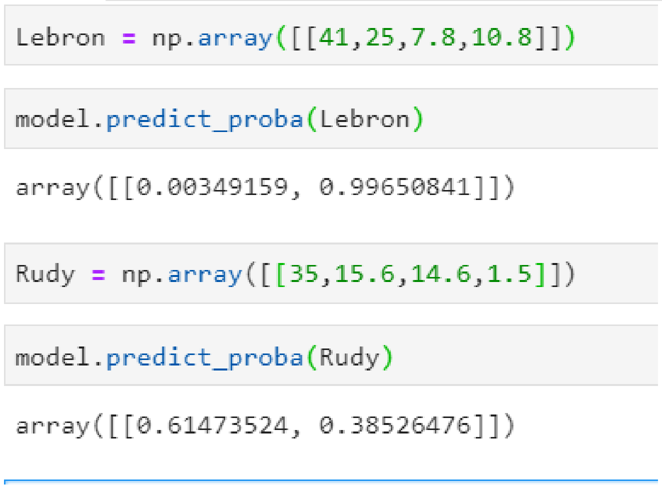
```
-1608.4757432145036
```

```python
X = np.hstack(( np.ones((488,1)), data[:,0:2] ))
beta = np.linalg.inv(X.T@X)@X.T@Y
488 * np.log( (Y-X@beta).T@(Y-X@beta)/488 ) + 2*(3)
```

```
-1606.8448004175168
```

```python
X = np.ones((488,1))
beta = np.linalg.inv(X.T@X)@X.T@Y
488 * np.log( (Y-X@beta).T@(Y-X@beta)/488 ) + 2*(1)
```

```
-1438.2812005201947
```



```python
Lebron = np.array([[41,25,7.8,10.8]])
```

```python
model.predict_proba(Lebron)
```

```
array([[0.00349159, 0.99650841]])
```

```python
Rudy = np.array([[35,15.6,14.6,1.5]])
```

```python
model.predict_proba(Rudy)
```

```
array([[0.61473524, 0.38526476]])
```