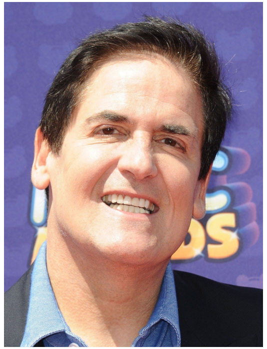
NBA Salary prediction: Linear Regression model and Web scraping

Calvin Yu

Introduction

How much should I pay Doncic?



- Mark Cuban (Owner of Dallas Mavericks) always want to invest the right amount of money to the right players.
- He decided to hire a data scientist to create a regression model that takes all the players' features to generate their predicted salary

Data

Basketball-Reference

ESPN

Methodology

- Request,BeautifulSoup : Web Scraping
- Pandas,Excel,numpy : Data cleaning,feature engineering
- Sklearn : Split the data into train/validation/test,and test different regression models(Linear Regression, LASSO,RIDGE, Polynomial(degrees=2),LASSO of Polynomial(degrees=2),RIDGE of Polynomial(degrees=2)
- Seaborn : Visualize the data
- Create a function that runs all the models on the given test set and give out the score of each model (see Appendix 1)

Results

- The columns of the best test set are ['Age', 'G', 'GS', 'MP', 'eFG%', 'FT%', 'TRB', 'AST', 'STL', 'BLK','TOV', 'PF', 'PTS', 'Year']

- kFold = 10

```
Linear Regression val R^2: 0.534
Linear Regression mean val R^2: 0.556:
Ridge Alpha Best_estimator : 0.100
Ridge Regression val R^2: 0.534
Ridge Regression mean val R^2: 0.556
Lasso Alpha Best_estimator : 0.100
Lasso Regression val R^2: 0.534
Lasso Regression mean val R^2: 0.558
Degree 2 polynomial regression val R^2: 0.596
Degree 2 polynomial Regression mean val R^2: 0.624
Degree 2 polynomial Ridge Alpha Best_estimator : 0.100
Degree 2 polynomial Ridge Regression val R^2: 0.622
Degree 2 polynomial Ridge Regression mean val R^2: 0.633
Degree 2 polynomial Lasso Alpha Best_estimator : 0.100
Degree 2 polynomial Lasso Regression val R^2: 0.621
Degree 2 polynomial Lasso Regression mean val R^2: 0.625
```

Result

Ridge of Polynomial (Degrees = 2)

R^2 score

Train = ~0.71

Validation = ~0.64

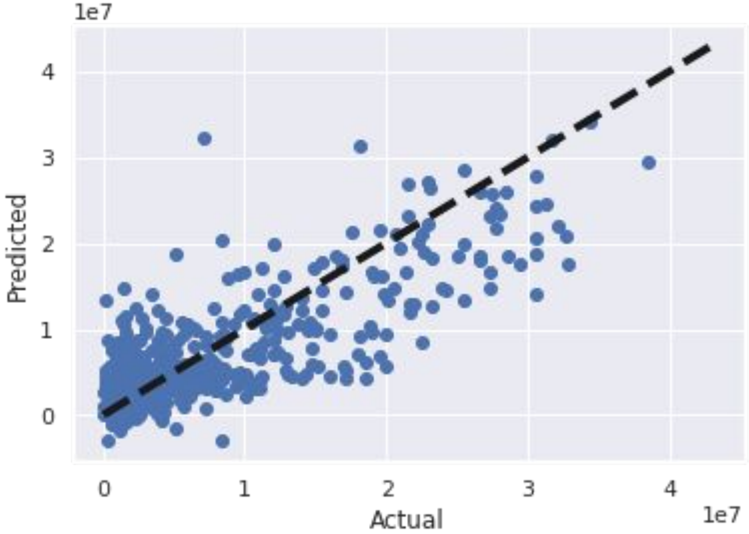
Test = ~0.64

Our model can explain ~64% of the variance (See appendix 2)

Unscale the coefficients and the intercept :

```
coefficients = np.true_divide(lm_reg_poly.coef_,  scaler.scale_)
lm_reg_poly.intercept_ - np.dot(coefficients, scaler.mean_)
```

Conclusion



The actual vs predicted plot:

- Our model tends to overestimate as the salaries goes up
- Need to acquire more data points or features to improve this model

Takeaways

- Learned how to use Beautifulsoup to scrape data online

- Learned how to split the data into train/validation/test and train them with different models

- The importance of R^2

Appendix

Appendix 1

```python
def split_and_validate(X,y):

    X, X_test, y, y_test = train_test_split(X, y, test_size=.2)
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=.25)

    kf = KFold(n_splits=10, shuffle=True)

    lm = LinearRegression()
    lm.fit(X_train, y_train)
    lm_cross_val_score = cross_val_score(lm, X_train, y_train, cv=kf, scoring='r2')

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train.values)
    X_val_scaled = scaler.transform(X_val.values)
    X_test_scaled = scaler.transform(X_test.values)


    ridge_grid_est = build_grid_search_est(Ridge(), X_train, y_train, cv=kf,
                                           alpha=np.logspace(-4, -1, 10))
    reg_alpha = ridge_grid_est.best_estimator_.alpha


    lm_reg = Ridge(alpha = reg_alpha)
    lm_reg.fit(X_train_scaled, y_train)
    lm_reg_cross_val_score = cross_val_score(lm_reg, X_train_scaled, y_train, cv=kf, scoring='r2')
    ridge_grid_est = build_grid_search_est(Ridge(), X_train, y_train, cv=kf,
                                           alpha=np.logspace(-4, -1, 10))



    lasso_grid_est = build_grid_search_est(Lasso(), X_train, y_train, cv=kf,
                                               alpha=np.logspace(-4, -1, 30))
    lasso_alpha = lasso_grid_est.best_estimator_.alpha
    lm_lasso = Lasso(alpha = lasso_alpha)
    lm_lasso.fit(X_train_scaled, y_train)
    lm_lasso_cross_val_score = cross_val_score(lm_lasso, X_train_scaled, y_train, cv=kf, scoring='r2')


    poly = PolynomialFeatures(degree=2)

    X_train_poly = poly.fit_transform(X_train)
    X_val_poly = poly.transform(X_val)
    X_test_poly = poly.transform(X_test)



    X_train_poly_scaled = scaler.fit_transform(X_train_poly)
    X_val_poly_scaled = scaler.fit_transform(X_val_poly)
    X_test_poly_scaled = scaler.fit_transform(X_test_poly)

    lm_poly = LinearRegression()
    lm_poly.fit(X_train_poly, y_train)
    lm_poly_cross_val_score = cross_val_score(lm_poly, X_train_poly, y_train, cv=kf, scoring='r2')


    ridge_grid_est_poly = build_grid_search_est(Ridge(), X_train_poly_scaled, y_train, cv=kf,
                                               alpha=np.logspace(-4, -1, 10))
    ridge_alpha_poly = ridge_grid_est_poly.best_estimator_.alpha
    lm_reg_poly = Ridge(alpha=ridge_alpha_poly)
    lm_reg_poly.fit(X_train_poly_scaled, y_train)
    lm_reg_poly_cross_val_score = cross_val_score(lm_reg_poly, X_train_poly_scaled, y_train, cv=kf, scoring='r2')


    lasso_grid_est_poly = build_grid_search_est(Lasso(), X_train_poly_scaled, y_train, cv=kf,
                                               alpha=np.logspace(-4, -1, 30))
    lasso_alpha_poly = lasso_grid_est_poly.best_estimator_.alpha
    lm_lasso_poly = Lasso(alpha = lasso_alpha_poly)
    lm_lasso_poly.fit(X_train_poly_scaled, y_train)
    lm_lasso_poly_cross_val_score = cross_val_score(lm_lasso_poly, X_train_poly_scaled, y_train, cv=kf, scoring='r2

    print(f'Linear Regression val R^2: {lm.score(X_val, y_val):.3f}')
    print(f'Linear Regression mean val R^2: {np.mean(lm_reg_cross_val_score):.3f}:')


    print(f'Ridge Alpha Best_estimator : {reg_alpha:.3f}')
    print(f'Ridge Regression val R^2: {lm_reg.score(X_val_scaled, y_val):.3f}')
    print(f'Ridge Regression mean val R^2: {np.mean(lm_reg_cross_val_score):.3f}')


    print(f'Lasso Alpha Best_estimator : {lasso_alpha:.3f}')
    print(f'Lasso Regression val R^2: {lm_lasso.score(X_val_scaled, y_val):.3f}')
    print(f'Lasso Regression mean val R^2: {np.mean(lm_lasso_cross_val_score):.3f}')

    print(f'Degree 2 polynomial regression val R^2: {lm_poly.score(X_val_poly, y_val):.3f}')
    print(f'Degree 2 polynomial Regression mean val R^2: {np.mean(lm_poly_cross_val_score):.3f}')


    print(f'Degree 2 polynomial Ridge Alpha Best_estimator : {ridge_alpha_poly:.3f}')
    print(f'Degree 2 polynomial Ridge Regression val R^2: {lm_reg_poly.score(X_val_poly_scaled, y_val):.3f}')
    print(f'Degree 2 polynomial Ridge Regression mean val R^2: {np.mean(lm_reg_poly_cross_val_score):.3f}')



    print(f'Degree 2 polynomial Lasso Alpha Best_estimator : {lasso_alpha_poly:.3f}')
    print(f'Degree 2 polynomial Lasso Regression val R^2: {lm_lasso_poly.score(X_val_poly_scaled, y_val):.3f}')
    print(f'Degree 2 polynomial Lasso Regression mean val R^2: {np.mean(lm_lasso_poly_cross_val_score):.3f}')
```

Appendix 2

```python
coefficients = np.true_divide(lm_reg_poly.coef_,  scaler.scale_)
```

```python
intercept = lm_reg_poly.intercept_ - np.dot(coefficients, scaler.mean_)
```

```python
print(list(zip(poly.get_feature_names(input_features= X.columns),coefficients)))
```

```
[('1', 0.0), ('Age', -34315102.95042527), ('G', 7105379.655350137), ('GS', -4715713.90573231), ('MP', -10058161.70
388103), ('eFG%', 17984213.671798367), ('FT%', 5433234.741548838), ('TRB', 32216044.306831583), ('AST', -109796953.
08034874), ('STL', -763629467.2376584), ('BLK', -491315503.55027676), ('TOV', -133934352.37789403), ('PF', 152618
7.85476118), ('PTS', -49761202.34958602), ('Year', 400028511.0725012), ('Age^2', -27494.135408815055), ('Age G', -
350.018033132781), ('Age GS', 3314.313544641842), ('Age MP', 489.59278643806823), ('Age eFG%', 1961.5938112851543),
('Age FT%', -3337.989660145404), ('Age TRB', 67025.08525031254), ('Age AST', 146855.1145203153), ('Age STL', 24847
9.41117265163), ('Age BLK', 205110.04833590257), ('Age TOV', -193078.49187997598), ('Age PF', -170009.43912907236),
('Age PTS', 45993.75739993591), ('Age Year', 17820.94121535213), ('G^2', 672.3649119122676), ('G GS', -2372.120817
51518), ('G MP', 9079.138467202763), ('G eFG%', 740.4557467787209), ('G FT%', 875.0252283499311), ('G TRB', -1078.
277250412177), ('G AST', -14245.497341423858), ('G STL', 595.6530108378442), ('G BLK', -4023.571572062281), ('G TO
', -4151.403344795251), ('G PF', 7163.66567954602), ('G PTS', -17876.24504272203), ('G Year', -3556.333945893538),
('GS^2', 1179.2197132890828), ('GS MP', -3547.2109582347503), ('GS eFG%', -1434.4996246812968), ('GS FT%', -1235.3
59042663671), ('GS TRB', 212.03801254757272), ('GS AST', 23891.158575823592), ('GS STL', -33964.88957450124), ('GS
BLK', 63536.23082392588), ('GS TOV', -2418.8033561465973), ('GS PF', -27799.229753595675), ('GS PTS', 8565.9349216
1714), ('GS Year', 2437.575060671144), ('MP^2', 4271.575696328759), ('MP eFG%', -11846.705097290556), ('MP FT%', -
551.350896700049), ('MP TRB', -17119.917009303565), ('MP AST', -33416.1901824925), ('MP STL', -172651.467510701),
('MP BLK', -76409.00730002331), ('MP TOV', 93804.35368596394), ('MP PF', -36061.022269664936), ('MP PTS', 14705.94
262818788), ('MP Year', 5400.7731174192195), ('eFG%^2', -700.3508668198862), ('eFG% FT%', 444.19861296316725), ('e
G% TRB', -7879.699517817193), ('eFG% AST', -24023.609848132535), ('eFG% STL', 26257.30441998883), ('eFG% BLK', -11
91.90772254803), ('eFG% TOV', 13050.993161524488), ('eFG% PF', 30762.860928309965), ('eFG% PTS', 15945.6319358426
5), ('eFG% Year', -8905.210653282174), ('FT%^2', -155.80418042152655), ('FT% TRB', 10789.094315432458), ('FT% AST',
34541.1038116518), ('FT% STL', -16812.68358505671), ('FT% BLK', -71888.9868257689), ('FT% TOV', -1891.17775726853
3), ('FT% PF', 20729.775391259216), ('FT% PTS', 7241.5912238303545), ('FT% Year', -2650.937499975228), ('TRB^2', 9
612.05245648722), ('TRB AST', 81056.37606405791), ('TRB STL', -30335.896434778784), ('TRB BLK', -137766.4235538760
3), ('TRB TOV', 329965.4037955466), ('TRB PF', -385210.22557265236), ('TRB PTS', -35701.10018272587), ('TRB Year',
-16714.758908526972), ('AST^2', -99931.59439168553), ('AST STL', 547806.5240379581), ('AST BLK', -276312.815669673
8), ('AST TOV', -117576.47917082915), ('AST PF', 594981.6000319666), ('AST PTS', -19644.78354307444), ('AST Year',
51850.072721113174), ('STL^2', 528669.5664332156), ('STL BLK', 1431816.2369645238), ('STL TOV', 308747.5668264959),
('STL PF', -723391.5173408098), ('STL PTS', 228247.76991940266), ('STL Year', 375801.77604104445), ('BLK^2', -1031
75.2600956709), ('BLK TOV', -30660.158996244194), ('BLK PF', 1121512.4582782765), ('BLK PTS', 144672.47096179743),
('BLK Year', 243271.24937686403), ('TOV^2', -279575.69880118483), ('TOV PF', -1093661.329285953), ('TOV PTS', -422
3.79117256057), ('TOV Year', 69072.83980591004), ('PF^2', -14694.47415873617), ('PF PTS', 78617.1372385233), ('PF
ear', -74399.43993163282), ('PTS^2', -3037.2300901891995), ('PTS Year', 23641.21826581479), ('Year^2', -99075.9668
3445)]
```

```python
intercept
```

```
-403810465586.6025
```

THANK YOU