Capstone 1 In-depth Analysis (Machine Learning) : Predicting MLB Players Salaries

In order to predict the salaries of MLB players, I chose to use several regression algorithms. Considering our target variable was continuous and our data had labels, it was best to use a supervised learning technique. I then needed to create a predictive model that best fit each of my player datasets. One of the key differences between the two datasets was the number of features used. A pitchers performance is measured using more metrics than that for a hitter. Thus leading to an increase in features.

Before building any machine learning models I had to prepare my dataset. The first issue I dealt with was scaling my data. Both datasets had values ranging from .100 to 1000+ . So it was important to have all the features within the same scale to avoid any adverse effect on the model. Both datasets were scaled using the Standard Scaler from Scikit-learn.

After scaling my data I then had to handle the categorical variables. In the data, the feature “positions” contained a range of nine positions that a player could play. To convert my categorical variables to numbers I used the pandas get dummies function. I also decided to use the log transformation of my target variable to limit the effect of any outliers. After completion of my preprocessing I began to build my models. The code and metrics can be seen below.

Machine Learning

The first regression model I built was using a DummyRegressor from scikit-learn. I used this model to get a baseline estimate for my metric of choice ‘Root Mean Squared Error’. The next algorithm I used was a Linear Regression model from Sci-kit learn. This model's performance was subpar, due to a lot of multicollinearity in my features. To correct this I used a dimensionality reduction technique known as Principal Component Analysis. Using PCA helped me reduce the number of features from 156 to 7, and the model's root mean squared error. The code can be seen below.

```
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

#drop useless columns
df_hitter_finall = df_hitter.drop(columns=['salary', 'borndate',
                                'Place', 'playerName', 'HT', 'teamName'], axis=1)
#convert categorical variables
dfh_obj = df_hitter_finall.select_dtypes(include=['object']).copy()
dfh_obj.columns
dummies = pd.get_dummies(df_hitter[['Bats', 'Throws', 'LeagueAbbr', 'posit']], drop_first=True)


# Drop the column with the independent variable (Salary), and columns for which we created
dummy variables
X_ = df_hitter_finall.drop(['Bats', 'Throws', 'LeagueAbbr', 'posit', 'adj_salary_filled' ],
axis=1).astype('float64')

# Define the feature set X and y.

X = pd.concat([X_, dummies], axis=1)
y = np.log(df_hitter_finall.adj_salary_filled)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#scale data
std = StandardScaler()
X_train_scaled = std.fit_transform(X_train)
X_test_scaled = std.transform(X_test)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=.20, random_state=42)


from sklearn.decomposition import PCA
pca = PCA(n_components=7)
model = LinearRegression()

X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

model.fit(X_train_pca, y_train)

y_pred = model.predict(X_test_pca)
y_pred_train = model.predict(X_train_pca)
rmse = np.sqrt(mean_squared_error(np.exp(y_test),np.exp(pred)))

print('Train r2 score: ', r2_score(y_pred_train, y_train))
print('Test r2 score: ', r2_score(y_test, y_pred))
```

```
print('Test RMSE:', rmse)

Train r2 score:  0.20153571282188076
Test r2 score:  0.5553069040901089
Test RMSE: 4644008.981008597
```

I then continued with other regression models, especially tree based ensemble methods. These models are known to be effective against outliers. A model robust to outliers was the Random Forest Regressor . I also made use of some of the best models currently available such as XGBoost, Gradient Boosting, LightGBM, HistGradientBoosing, and CatBoost. In order to find the right combination of hyperparameters for each model I used Grid Search or Randomized Search.

The best performing model for the hitter dataset was the LightGBM Regressor. This model's final metrics was an R-squared of 0.80 and RMSE of 3,236,633. The hyperparameters were 'n_estimators' which were set to 200 and 'max_depth' set to 6. The code for the model with the best hyperparameters, as well as the predictions can be seen below.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

#drop useless columns
df_hitter_finall = df_hitter.drop(columns=['salary', 'borndate',
                                'Place', 'playerName', 'HT', 'teamName'], axis=1)
#encode categorical features
dfh_obj = df_hitter_finall.select_dtypes(include=['object']).copy()
dfh_obj.columns
dummies = pd.get_dummies(df_hitter[['Bats', 'Throws', 'LeagueAbbr', 'posit']], drop_first=True)


# Drop the column with the independent variable (Salary), and columns for which we created
dummy variables
X_ = df_hitter_finall.drop(['Bats', 'Throws', 'LeagueAbbr', 'posit', 'adj_salary_filled' ],
axis=1).astype('float64')
```

```
# Define the feature set X, y
X = pd.concat([X_, dummies], axis=1)
y = df_hitter_finall.adj_salary_filled
y_log = np.log(df_hitter_finall.adj_salary_filled)


X_train, X_test, y_train, y_test = train_test_split(X, y_log, test_size=0.2, random_state=42)

#scale data
std = StandardScaler()
X_train_scaled = std.fit_transform(X_train)
X_test_scaled = std.transform(X_test)

from lightgbm import LGBMRegressor

lgbm = LGBMRegressor(n_estimators=200, max_depth=6, num_leaves=15)
lgbm.fit(X_train_scaled, y_train)
y_pred = lgbm.predict(X_test_scaled)
y_pred_train_ = lgbm.predict(X_train_scaled)

rmse = np.sqrt(mean_squared_error(np.exp(y_test), np.exp(y_pred)))

print(rmse)
print('Train r2 score: ', r2_score(y_pred_train_, y_train))
print('Test r2 score: ', r2_score(y_test, y_pred))

3236633.8627096354
Train r2 score:  0.8753271859626053
Test r2 score:  0.8003821458146158
```

	Predicted	Actual
1375	19076500.5	22215589.6
1376	562280.9	563213.0
1377	675649.4	557998.1
1378	640047.2	555556.3
1379	6832035.1	5399622.4
1380	619921.2	555000.0
1381	532092.0	540592.3
1382	491367.7	534488.5
1383	8945658.7	2969792.3
1384	17156686.0	12959093.7

In the pitcher dataset the best model was the CatBoost Regressor. This model's final metrics was an R-squared of 0.77 and RMSE of 2,754,297. The hyperparameters 'learning rate' were set to 0.03 and 'iterations' to 3000. I also implemented the use of early stopping rounds in this model to minimize any possible overfitting.
Working with this model was rather interesting because it encodes any categorical variables in the dataset. The code for the model with the best hyperparameters, as well as the predictions can be seen below.

```
from catboost import CatBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

df_pitcher_fin = df_pitcher.drop(columns=['salary', 'borndate',
                            'Place', 'playerName', 'HT', 'teamName'], axis=1)


X = df_pitcher_fin.drop(columns=['adj_salary_filled'])
y = df_pitcher_fin.adj_salary_filled
y_log = np.log(df_pitcher_fin.adj_salary_filled)

X_train, X_test, y_train, y_test = train_test_split(X, y_log, test_size=0.2, random_state=42)


cat = CatBoostRegressor(iterations=3000, verbose=200,
                    learning_rate=0.03,loss_function='RMSE',
                    early_stopping_rounds= 200,
```

```
                    random_seed=42)
cat.fit(X_train, y_train, cat_features=['Bats', 'Throws', 'LeagueAbbr', 'posit'],  eval_set=(X_test, y_test)
      )
y_pred = cat.predict(X_test)
y_pred_train = cat.predict(X_train)
rmse = np.sqrt(mean_squared_error(np.exp(y_test), np.exp(y_pred)))
print('Train r2 score: ', r2_score(y_pred_train, y_train))
print('Test r2 score: ', r2_score(y_test, y_pred))
print(rmse)
Train r2 score:  0.8225891937818299
Test r2 score:  0.7773951221427647
2754297.969240993
```

	predicted	actual
0	485164.3	468974.9
1	575820.0	557998.1
2	2904430.9	1670276.7
3	551642.9	559953.7
4	542355.0	554875.3
5	560384.2	557998.1
6	479937.5	468974.9
7	632007.9	566583.6
8	1921405.9	556580.4
9	1572400.7	5025000.0

After looking into each model's individual feature importance attribute, I was able to uncover some interesting results. There was only one model that had ranked the feature 'positions' within its top 10 of feature importance. This model was the CatBoost Regressor used in the hitter dataset. Surprisingly, other models did not view this feature as informative enough. From a general standpoint a players position is thought of to be an important factor in their salary.

Overall I was able to get over a 40% reduction in the root mean squared error of the baseline models in the hitter and pitcher datasets respectively. For future iterations I will gather data on player injuries and medical history. Collecting information on a player's past injuries will help predict any future injuries as well as affect their total value. Therefore leading to a more precise salary prediction. Another feature that could

be added is a player's place of origin. This could be within the states or at the international level, as they each get paid different bonuses.

Final Metrics

- **Hitter Dataset**

Models	RMSE	R2
Baseline Model	$5,397,560	0.00
Linear Regression w PCA	$4,644,008	0.55
RandomForest Regressor	$3,510,004	0.78
XGBoost Regressor	$3,315,287	0.80
HistGradientBoosting Regressor	$3,293,081	0.79
CatBoost Regressor	$3,272,132	0.80
LightGBM Regressor	$3,236,633	0.80

- **Pitcher Dataset**

Models	RMSE	R2
Baseline Model	$4,683,385	0.00
XGBoost Regressor	$2,832,288	0.76
LightGBM Regressor	$2,818,405	0.76
RandomForest Regressor	$2,796,006	0.77
HistGradientBoost Regressor	$2,757,535	0.77
CatBoost Regressor	$2,754,287	0.77