Baseball Performance Prediction Project

Overview

Project Overview

This project focuses on predicting baseball player performance using machine learning techniques. The goal is to develop a model that can forecast a player's future performance in terms of WAR, based on their historical statistics. I am a massive fan of Baseball and wanted to learn more about data science, machine learning and statistical models and so chose to pursue this project.

Baseball

Baseball is a popular sport played between two teams, each consisting of nine players. The primary objective is to score runs by hitting a thrown ball with a bat and running around a series of four bases arranged in a diamond shape. Baseball is known for its rich history, complex rules, and extensive statistical analysis, making it a prime candidate for data-driven projects.

Wins Above Replacement (WAR)

Wins Above Replacement (WAR) is a statistical metric used in baseball to measure a player's overall contribution to their team's success. It quantifies how many more wins a player contributes to their team compared to a hypothetical replacement-level player, typically defined as a freely available player from the minor leagues or waiver wire. WAR takes into account a player's offensive, defensive, and pitching contributions, providing a comprehensive assessment of their value to the team.

Steps Involved

1. **Data Collection**: Batting statistics data from 2002 to 2023 is retrieved using the pybaseball library. This dataset includes various metrics such as hits, home runs, strikeouts, and WAR for each player in each season.
2. **Data Preprocessing**: The retrieved data undergoes preprocessing steps to clean and prepare it for analysis. This includes removing players with insufficient data, handling missing values, and encoding categorical variables.
3. **Feature Selection and Engineering**: Feature selection techniques, such as Ridge regression and Sequential Feature Selector, are applied to select the most relevant features for modeling. Additionally, new features are engineered, such as player performance history and group average WAR, to enhance the predictive power of the model.
4. **Model Evaluation**: The model's performance is evaluated through backtesting, where the model is trained on historical data and tested on future seasons. Mean squared error is used as a metric to assess the model's accuracy in predicting WAR.
5. **Results Analysis**: The results of the model are analyzed to gain insights into player performance and the factors influencing it. This includes examining player performance

statistics, displaying coefficients of the Ridge regression model, and comparing actual and predicted WAR for individual players.

By following these steps, the project aims to provide valuable insights into baseball player performance and contribute to the growing field of sports analytics. This project was developed with inspiration and learning from the following tutorial. Dataquest YouTube Video

In [ ]:
```
import os
import pandas as pd
import numpy as np
from pybaseball import batting_stats
```

Data Collection

In [ ]:
```
START = 2002
END = 2023
```

In [ ]:
```
# Fetch batting statistics from 2002 to 2023 with a minimum qualification of 200 plat
batting = batting_stats(START, END, qual=200)
```

In [ ]:
```
batting.to_csv("batting.csv")
```

Data Preprocessing

In [ ]:
```
# Filter out players with only one season of batting statistics
batting = batting.groupby("IDfg", group_keys=False).filter(lambda x: x.shape[0] > 1)
```

In [ ]:
```
# Define a function to add the next season's Wins Above Replacement (WAR) for each pl
def next_season(player):
    player = player.sort_values("Season")
    player["Next_WAR"] = player["WAR"].shift(-1)
    return player
```

In [ ]:
```
batting = batting.groupby("IDfg", group_keys=False).apply(next_season)
```

In [ ]:
```
# Count the number of null values in each column
null_count = batting.isnull().sum()
```

In [ ]:
```
# Get the names of columns with no null values
complete_cols = list(batting.columns[null_count == 0])
```

In [ ]:
```
# Select only the complete columns along with the "Next_WAR" column
batting = batting[complete_cols + ["Next_WAR"]].copy()
```

In [ ]:
```
# Display data types of columns that are of object type
batting.dtypes[batting.dtypes == "object"]
```

Out[ ]:
```
Name       object
Team       object
Dol        object
Age Rng    object
dtype: object
```

In [ ]:
```
# Remove irrelevant columns
del batting["Dol"]
del batting["Age Rng"]
```

In [ ]:
```
# Convert the "Team" column to categorical and encode it
batting["team_code"] = batting["Team"].astype("category").cat.codes
```

In [ ]:
```
# Create a copy of the batting dataframe and remove rows with null values
batting_full = batting.copy()
batting = batting.dropna().copy()
```

Feature Selection and Engineering

In [ ]:
```
from sklearn.linear_model import Ridge
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import TimeSeriesSplit
```

In [ ]:
```
# Initialize Ridge regression model
# Ridge regression is chosen as the modeling technique due to its ability to handle m
rr = Ridge(alpha=1)

# Define TimeSeriesSplit for cross-validation
# TimeSeriesSplit is used for cross-validation to ensure that temporal dependencies a
split = TimeSeriesSplit(n_splits=3)

# Initialize Sequential Feature Selector
# Sequential Feature Selector is employed to automatically select the most relevant f
sfs = SequentialFeatureSelector(rr, n_features_to_select=20, direction="forward", cv=
```

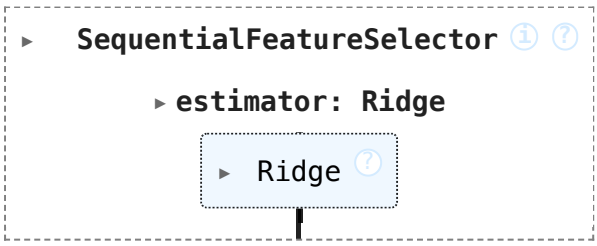
In [ ]:
```
# Define removed columns and selected columns for feature selection
removed_columns = ["Next_WAR", "Name", "Team", "IDfg", "Season"]
selected_columns = batting.columns[~batting.columns.isin(removed_columns)]
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [ ]:
```
# Scale selected columns using MinMaxScaler
scaler = MinMaxScaler()
batting.loc[:, selected_columns] = scaler.fit_transform(batting[selected_columns])
```

In [ ]:
```
# Perform feature selection using Sequential Feature Selector
sfs.fit(batting[selected_columns], batting["Next_WAR"])
```

Out[ ]:



Model Building and Evaluation

In [ ]:
```
# Get selected predictors
predictors = list(selected_columns[sfs.get_support()])
```

In [ ]:
```
# Define a function for backtesting the model
def backtest(data, model, predictors, start=5, step=1):
    all_predictions = []
    years = sorted(data["Season"].unique())

    for i in range(start, len(years), step):
        current_year = years[i]
```

```
        train = data[data["Season"] < current_year]
        test = data[data["Season"] == current_year]

        model.fit(train[predictors], train["Next_WAR"])
        preds = model.predict(test[predictors])
        preds = pd.Series(preds, index=test.index)
        combined = pd.concat([test["Next_WAR"], preds], axis=1)
        combined.columns = ["actual", "prediction"]

        all_predictions.append(combined)

    return pd.concat(all_predictions)
```

In [ ]:
```
# Perform backtesting and get predictions
predictions = backtest(batting, rr, predictors)
```

Results Analysis

In [ ]:
```
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
# Calculate mean squared error
mean_squared_error(predictions["actual"], predictions["prediction"])
```

Out[ ]: 2.754970615164269

In [ ]:
```
# Define a function to calculate player's WAR history
def player_history(df):
    df = df.sort_values("Season")

    df["player_season"] = range(0, df.shape[0])
    df["war_corr"] = list(df[["player_season", "WAR"]].expanding().corr().loc[(slice(
    df["war_corr"].fillna(1, inplace=True)

    df["war_diff"] = df["WAR"] / df["WAR"].shift(1)
    df["war_diff"].fillna(1, inplace=True)

    df["war_diff"][df["war_diff"] == np.inf] = 1

    return df
```

In [ ]:
```
batting = batting.groupby("IDfg", group_keys=False).apply(player_history)
```

In [ ]:
```
# Define a function to calculate group average WAR for each season
def group_average(df):
    return df["WAR"] / df["WAR"].mean()
```

In [ ]:
```
# Calculate and add group average WAR for each season
batting["war_season"] = batting.groupby("Season", group_keys=False).apply(group_avera
```

In [ ]:
```
# Define new predictors including player's season, WAR correlation, group average WAR
new_predictors = predictors + ["player_season", "war_corr", "war_season", "war_diff"]
```

In [ ]:
```
# Perform backtesting with new predictors
predictions = backtest(batting, rr, new_predictors)
```

In [ ]:
```
# Calculate mean squared error with new predictors
mean_squared_error(predictions["actual"], predictions["prediction"])
```

Out[ ]: 2.6730043982617815

In [ ]:
```
# Calculate the difference between actual and predicted WAR
diff = predictions["actual"] - predictions["prediction"]
```

In [ ]:
```
# Merge predictions with the batting dataframe
merged = predictions.merge(batting, left_index=True, right_index=True)
```

In [ ]:
```
# Add absolute difference between actual and predicted WAR to the merged dataframe
merged["diff"] = (predictions["actual"] - predictions["prediction"]).abs()
```

In [ ]:
```
# Display relevant columns sorted by absolute difference
merged[["IDfg", "Season", "Name", "WAR", "Next_WAR", "diff"]].sort_values(["diff"])
```

Out[ ]:

	IDfg	Season	Name	WAR	Next_WAR	diff
4295	9345	2014	Brock Holt	0.347826	2.0	0.000139
5370	4922	2014	Ender Inciarte	0.329193	2.4	0.000221
5608	1177	2018	Albert Pujols	0.180124	-0.6	0.000612
2563	6035	2009	David Murphy	0.291925	1.8	0.000879
2712	5933	2021	Jean Segura	0.422360	1.8	0.002685
...	...	...	...	...	...	...
3388	5631	2010	Matt Kemp	0.211180	8.3	6.406338
3299	4810	2007	Brian McCann	0.304348	8.6	6.479917
889	9166	2010	Buster Posey	0.459627	10.1	6.751022
2629	11579	2014	Bryce Harper	0.310559	9.3	7.438709
464	15640	2021	Aaron Judge	0.559006	11.6	7.619577

4409 rows × 6 columns

Thank You

Final Note: Thank you for reviewing the code! To note - there are several areas where the model could be improved, these include but are not limited to:

1. **Feature Selection Techniques**: Explore alternative feature selection methods beyond Sequential Feature Selector, such as Recursive Feature Elimination or Principal Component Analysis, to identify the most relevant features for modeling.

2. **Machine Learning Algorithms**: Experiment with different machine learning algorithms beyond Ridge regression. Techniques like Random Forests, Gradient Boosting Machines, or Neural Networks could potentially provide better predictive performance.

3. **Incorporating Additional Data Sources**: Consider integrating additional data sources, such as player injury history, weather conditions during games, or advanced player metrics, to enrich the model's understanding of player performance and improve its predictive accuracy.