RATINGS REGRESSION

DATE: *03/15/24*

AUTHOR- *PINAK SHOME*

AFFILIATION- *UNIVERSITY OF DENVER*

INSTRUCTOR- *DON DALTON*

COURSE- *DATA SCIENCE TOOLS 2*

COURSE NUMBER- *4448*

DEPARTMENT- *COMPUTER SCIENCE*

INTRODUCTION

RESEARCH SIGNIFICANCE

This paper explores the intricate details of how machine learning models differ in making predictions when the dataset used has characteristics which are partially (but not completely) favorable to all the models being implemented. Since a soccer database is being used here, it has significance in understanding how analytics of different characteristics can have an impact on the game. But mainly, this paper aims to provide a concise a model selection recommendation when the features of the dataset don't clearly fit the criteria of any specific model. It provides insights into what models would be most suited for a given task with computational and metric prediction accuracy constraints in mind.

DATASET DESCRIPTION

Taken from https://www.kaggle.com/datasets/hugomathien/soccer(10k rows after cleaning - Mid Sized). This was originally a sql database file. I've extracted two tables of all available for this case study.

```
df_Player.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11060 entries, 0 to 11059
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  11060 non-null  int64
 1   player_api_id       11060 non-null  int64
 2   player_name         11060 non-null  object
 3   player_fifa_api_id  11060 non-null  int64
 4   birthday            11060 non-null  object
 5   height              11060 non-null  float64
 6   weight              11060 non-null  int64
dtypes: float64(1), int64(4), object(2)
memory usage: 605.0+ KB
```

```
Int64Index: 180354 entries, 0 to 183977
Data columns (total 42 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   id                   180354 non-null  int64
 1   player_fifa_api_id   180354 non-null  int64
 2   player_api_id        180354 non-null  int64
 3   date                 180354 non-null  object
 4   overall_rating       180354 non-null  float64
 5   potential            180354 non-null  float64
 6   preferred_foot       180354 non-null  object
 7   attacking_work_rate  180354 non-null  object
 8   defensive_work_rate  180354 non-null  object
 9   crossing             180354 non-null  float64
 10  finishing            180354 non-null  float64
 11  heading_accuracy     180354 non-null  float64
 12  short_passing        180354 non-null  float64
 13  volleys              180354 non-null  float64
 14  dribbling            180354 non-null  float64
 15  curve                180354 non-null  float64
 16  free_kick_accuracy   180354 non-null  float64
 17  long_passing         180354 non-null  float64
 18  ball_control         180354 non-null  float64
 19  acceleration         180354 non-null  float64
 20  sprint_speed         180354 non-null  float64
 21  agility              180354 non-null  float64
 22  reactions            180354 non-null  float64
 23  balance              180354 non-null  float64
 24  shot_power           180354 non-null  float64
 25  jumping              180354 non-null  float64
 26  stamina              180354 non-null  float64
 27  strength             180354 non-null  float64
 28  long_shots           180354 non-null  float64
 29  aggression           180354 non-null  float64
 30  interceptions        180354 non-null  float64
 31  positioning          180354 non-null  float64
 32  vision               180354 non-null  float64
 33  penalties            180354 non-null  float64
 34  marking              180354 non-null  float64
 35  standing_tackle      180354 non-null  float64
 36  sliding_tackle       180354 non-null  float64
 37  gk_diving            180354 non-null  float64
 38  gk_handling          180354 non-null  float64
 39  gk_kicking           180354 non-null  float64
```

The general trend of this dataset is:

All higher value float64 variables contribute to a higher target variable rating.

Anomaly: There are several instances of rows where the target variable values are higher even though most of their 'float64' magnitudes are lower, which tell us that the

features are not necessarily strong indicators of the target. This makes it suitable for analyzing how models would learn on datasets with no clear patterns and help us understand the nuances of how we can leverage model architectures when there is no clear-cut trends present.

PREPROCESSING

DATA PREPARATION

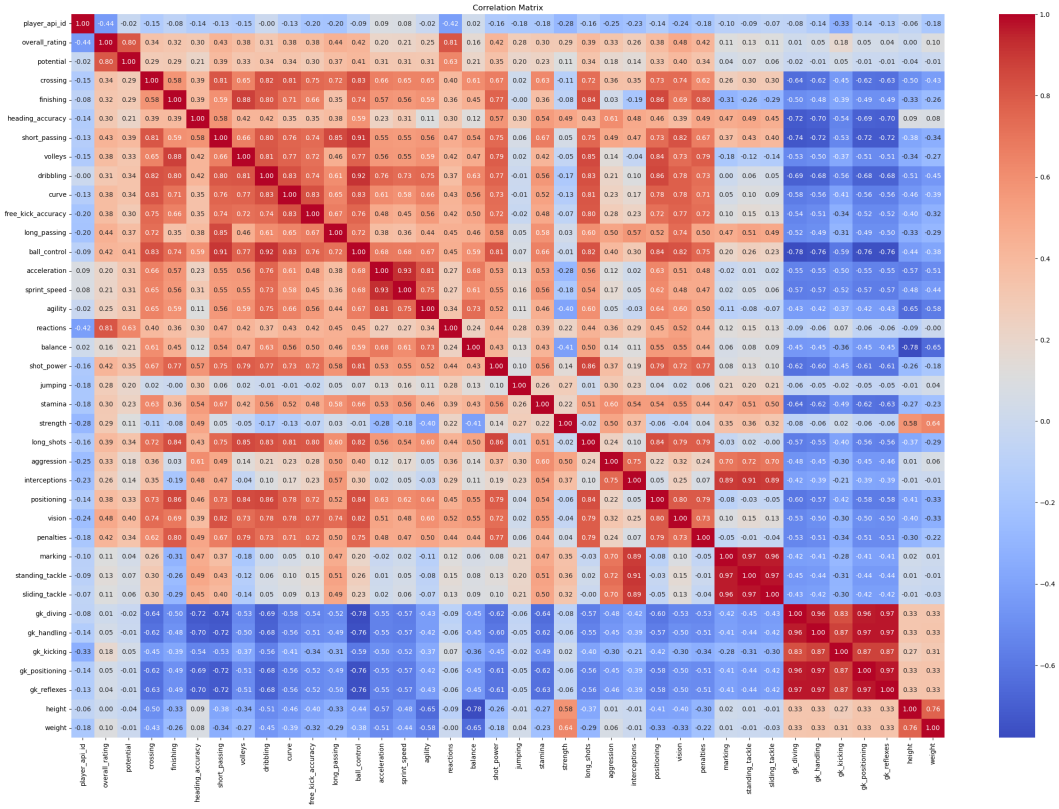
1) Dropped Null value column rows

2) Mean imputed float64 variables for repeating ‘id’ instances and mode imputed categorical variables. Reduced categorical variables’ range based on frequency distribution.

3) Converted ‘date’ variable to age (Since higher magnitude generally points to a higher target magnitude)

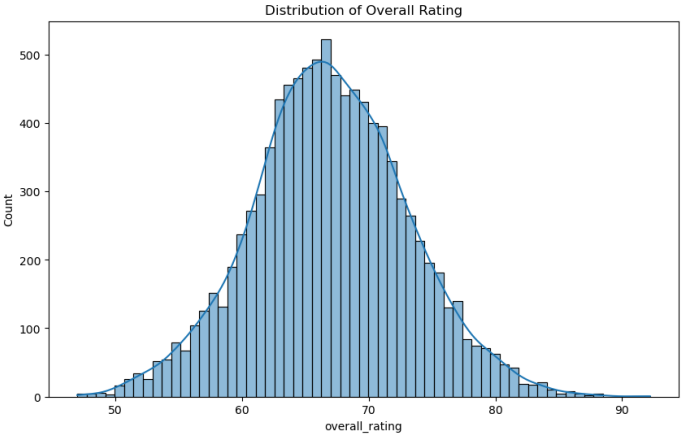
4) Left join on ‘player_id’ column

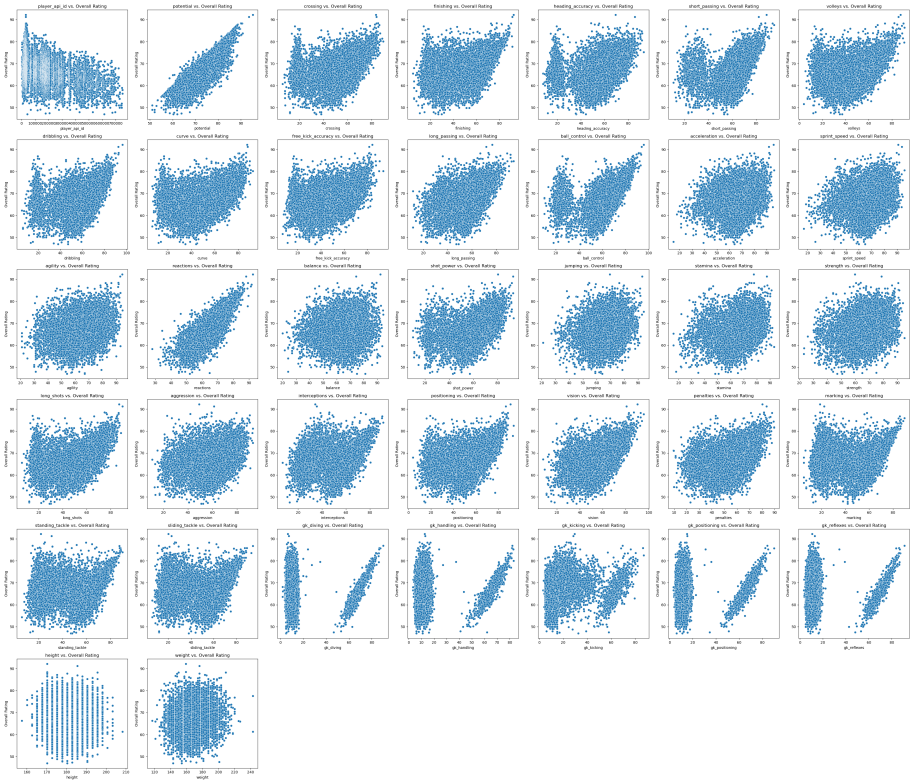
EDA&VISUALIZATION

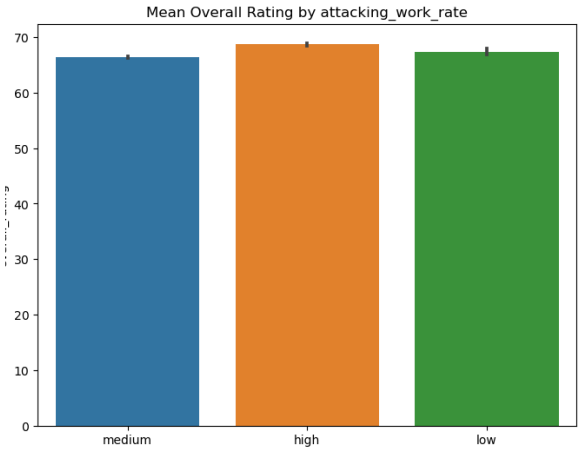


OBSERVATIONS:

There is significant linearity present between some features and target variable. There’s also significant collinearity present.







MODELLING

Model Workflow Summary: Picked three variations of linear models (Linear, Lasso and Ridge) pertaining to the fact that the dataset displayed linearity between features and the target variable. Since there was non-linearity and complex relationships present as well (Referencing correlation matrix and Pearson-correlation values), I've additionally picked the random forest regressor and lastly neural nets to compare how well random forest is capturing the dynamics of relationships between all the variables (Given Feature Irrelevance). Since this dataset isn't very high dimensional with over-complexity and has features that truly do not contribute to the target variable along with some complex relationships between features, Random forest may not perform significantly better than linear models. We'll use neural networks to have a benchmark of Random Forest' performance.

```python
X = df.drop(columns=['player_api_id', 'overall_rating'])
y = df['overall_rating']
num_cols = X.select_dtypes(include=['float64', 'int64']).columns
cat_cols = X.select_dtypes(include=['object']).columns
# Define the preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso()}
# Dictionary to store MSE for each model
mse_scores = {}
# Loop through models, fit, predict, and compute MSE
for name, model in models.items():
    # Append regressor to preprocessing pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('regressor', model)])

    # Fit the model
    pipeline.fit(X_train, y_train)

    # Predict
    y_pred = pipeline.predict(X_test)

    # Compute and store MSE
    mse_scores[name] = mean_squared_error(y_test, y_pred)
    print(f"{name} MSE: {mse_scores[name]}")
```

```
Linear Regression MSE: 3.4097614265957508
Ridge Regression MSE: 3.4097991818676148
Lasso Regression MSE: 6.504610283467142
```

```python
from sklearn.metrics import mean_squared_error
import numpy as np

# Calculate the mean of the target variable in the training set
mean_train = y_train.mean()

# Create an array filled with the mean value, with the same shape as y_tes
baseline_predictions = np.full(shape=y_test.shape, fill_value=mean_train)

# Compute the MSE for the baseline predictions
baseline_mse = mean_squared_error(y_test, baseline_predictions)
print(f"Baseline Model (Mean) MSE: {baseline_mse}")
```

```
Baseline Model (Mean) MSE: 37.73637259911164
```

```python
from sklearn.model_selection import GridSearchCV
# Define the parameter grid to search
param_grid = {
    'regressor__n_estimators': [100, 200, 300],
    'regressor__max_depth': [None, 10, 20, 30],
}
# Initialize the GridSearchCV object
grid_search = GridSearchCV(model_rf, param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)
# Fit it to the training data
grid_search.fit(X_train, y_train)

# Print the best parameters and the corresponding score
print("Best parameters:", grid_search.best_params_)
print("Best MSE (negative):", grid_search.best_score_)

# Predict on the test set using the best found parameters
y_pred_best = grid_search.predict(X_test)

# Compute the MSE for the predictions
mse_best = mean_squared_error(y_test, y_pred_best)
print(f"Best Random Forest Regressor MSE: {mse_best}")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters: {'regressor__max_depth': 20, 'regressor__n_estimators': 300}
Best MSE (negative): -2.0139939350235743
Best Random Forest Regressor MSE: 1.82965912830981
```

Model	MSE
Baseline Model (Mean)	37.736373
Linear Regression	3.409761
Ridge Regression	3.409799
Lasso Regression	6.504610
Random Forest Regressor	1.841383
Random Forest Regressor (Hyperparameter Tuned)	1.829659
Neural-Nets	0.634445

CONCLUSION

1) Base Linear regression and Ridge Regression perform very similar to random forest. Neural Nets perform best due to their ability to learn patterns through their hidden layers even if there is a high degree of feature irrelevance.

2) Lasso Regression performs poorly compared to the other linear models because of its feature elimination process in a dataset with low feature relevance.

3) There is a good deal of linearity in the dataset, which explains the enhanced performance of linear models compared to base-mean model.

4) Random Forest does not outperform the linear models due to feature irrelevance present, along with lack of very high dimensionality and dataset size which prevents it from learning the dataset optimally.

RECOMMENDATIONS

1) For small datasets with mid to low dimensionality with significant presence of linearity even with collinear features existing, linear models should be preferred over more complex models if a small margin error of accuracy is acceptable.

2) For high dimensional datasets with complex features relationships, more complex models like Random Forest and Neural Nets should be prioritized.

3) For large datasets with high dimensionality and feature irrelevance neural networks do a very good of learning a pattern and optimizing predictions.

4) Neural Networks are also significantly more computationally expensive as compared to RF models, so unless feature irrelevance is very high, RF could still give us very similar performance.

5) For very sparse datasets, Neural Networks will outperform Random Forest models and hence should be prioritized if complex models are being used

6) For small datasets, complex models do not generalize well beyond training data, so if scoring metrics are not a very huge concern and computing resources are limited, simpler linear models make a better choice.