MATH 596 FINAL PROJECT
SOCCER PLAYER SCOUTING WITH PCA AND GA

PROFESSOR VAN VLECK
TRI PHAM

Introduction

Upon winter transfer window, football clubs are looking to buy good quality players with affordable price to improve their team performance for the remaining season, whether for the title race or avoiding relegation. This project will look at applying different techniques and models used in data science and machine learning to help scout potential players that fit with a club's budget. By looking at different attributes belonging to a player, such as value, wage, age, international caps, and many more features, we can predict how the player would fit in a club's squad and budget. We will be looking at the default dataset of FIFA 20 from kaggle [1]. The ultimate goal is to build a sound scouting system that could help scouting agents recruit the most potential players to help the club's overall campaign.

Background

This project assumes an understanding of Principle Component Analysis (PCA). Additionally, I will be attaching a pseudocode for the Genetic Algorithm to assist my explanation. First, PCA is a tool to help reduce the dimensions of the dataset with minimal loss of information. The goal of PCA is to project a feature space onto a smaller subspace that stills represents the data effectively. As described in the textbook "Mathematics for Machine Learning" [2], we follow 4 main procedures.

Date: December 12, 2020.

1) We begin with centering the data by computing the mean μ of the dataset and subtracting it from every single data point. This step is called mean subtraction.

$$x_i - \frac{1}{n}\sum_{i=1}^{n} x_i$$

2) Next we divide the data points by the standard deviation σ_d of the dataset for every dimension $d = 1, ..., D$. This step is called the standardization of the data, with $x^{(d)} - \mu_d$ being the mean subtraction where $x^{(d)}$ is the d-th component of x and μ_d is the mean.

$$\frac{x^{(d)} - \mu_d}{\sigma_d}$$

3) Then, compute the data covariance matrix and its eigenvalues and corresponding eigenvectors. This step is called the eigendecomposition of the covariance matrix.

$$cov(x, y) = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{n - 1}$$

The covariance matrix for a set of data with n-dimension. Obtaining the covariance matrix, we can decompose it to find the eigenvalues and corresponding eigenvectors. [3]

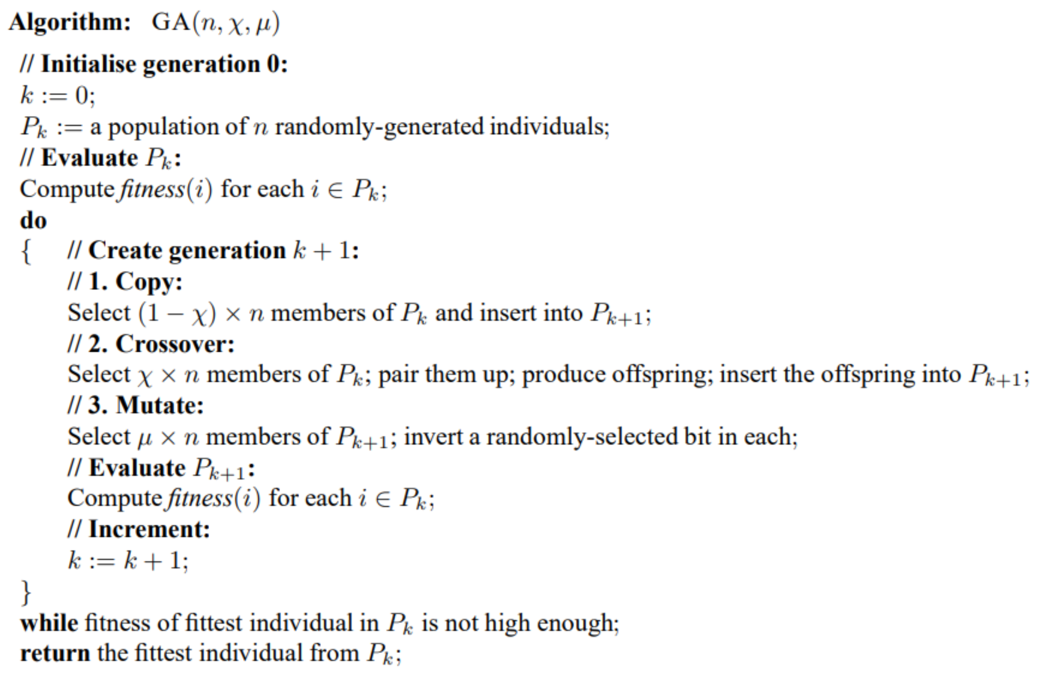
$$C^{(n\times n)} = (c_{i,j}|c_{i,j} = cov(i^{th}\ dimension, j^{th}\ dimension))$$

4) Having done the above steps, we can project any data point $x_* \in R^D$ onto the principal subspace. The projection will be

$$\hat{x_*} = BB^T x_*\ with\ z_* = B^T x_*$$

Here coordinates z_* are with respect to the basis of the principle subspace and B is the matrix that contains the eigenvectors that are associated with the largest eigenvalues of the data covariance matrix as columns.

Genetic Algorithm is an iterative process, and each iteration attempts to find the best population of individuals. The individuals are evaluated using a fitness function. A new generation of the population is obtained by mutation to find fitter individuals from the current generation. Some of these individuals admitted to the next generation remain unchanged while others go through crossover and mutation to create new offspring. Following the pseudocode, n is the number of individuals in the population; χ is the fraction of the population to be replaced by crossover in each iteration; and μ is the mutation rate. [4]



```
Algorithm: GA($n, \chi, \mu$)
 // Initialise generation 0:
 $k := 0$;
 $P_k :=$ a population of $n$ randomly-generated individuals;
 // Evaluate $P_k$:
 Compute fitness($i$) for each $i \in P_k$;
 do
 {    // Create generation $k + 1$:
      // 1. Copy:
      Select $(1 - \chi) \times n$ members of $P_k$ and insert into $P_{k+1}$;
      // 2. Crossover:
      Select $\chi \times n$ members of $P_k$; pair them up; produce offspring; insert the offspring into $P_{k+1}$;
      // 3. Mutate:
      Select $\mu \times n$ members of $P_{k+1}$; invert a randomly-selected bit in each;
      // Evaluate $P_{k+1}$:
      Compute fitness($i$) for each $i \in P_k$;
      // Increment:
      $k := k + 1$;
 }
 while fitness of fittest individual in $P_k$ is not high enough;
 return the fittest individual from $P_k$;
```

Fig1. A version of GA's pseudocode

Implementation

Data

The data of players are provided by FIFA 20 complete dataset from Kaggle [1]. The original dataset includes 18,278 samples (players in professional soccer league) with 104 different attributes belonging to that player. An initial plan was to implementing a web scraper to retrieve the players' most updated data for real-time resemblance. Unfortunately, this remains the goal for future implementation since websites are still constantly updating their content which makes it difficult the scrap data without getting halted with parsing errors.

Data cleaning and extraction

Even though the csv file from Kaggle is properly formatted, there was still a problem when I downloaded it. Particularly, the names that have letters not belonging to the American alphabet was not translated properly, but this was later fixed with manual copying and pasting. As I have a large number of players, I decided to drop those that have symbolic attributes such as pictures, body type, and a few others. After a selective process, I ended up with 16,980 players and 54 attributes (dataset *).

Working on Principle Component Analysis implementation, since it results in a feature subspace that maximizes the variance along the axes, standardizing the data is beneficial. The StandardScaler provides both mean subtraction and standardization functionality. Next, I worked on the eigendecomposition of the covariance matrix. Essentially, I would compute the eigenvalues and their corresponding eigenvectors after computing the covariance matrix [2][5]. Then, I calculated the proportion of variance (PoV) to determine a good number of transformed attributes. The PoV tells us how much weight a group of eigenvalues has when comparing to the total eigenvalues. Usually, a good PoV is $\geq .90$. I managed to pick out the first 20 eigenvalues and their corresponding eigenvectors for my transformed attributes.

```
PoV's:  [ 37.94  47.72  56.22  59.98  62.97  65.43  66.76  68.77  70.76  72.73
  74.68  76.61  78.54  80.39  82.3   84.2   86.08  87.96  89.83  90.92
  91.74  92.48  93.11  93.73  94.24  94.75  95.23  95.64  96.04  96.42
  96.78  97.12  97.44  97.75  97.76  98.03  98.26  98.48  98.53  98.58
  98.63  98.69  98.75  98.82  99.02  99.2   99.37  99.52  99.63  99.75
  99.87 100.  ]
```

Working on Genetic Algorithm (GA) implementation, I have an initial population containing five sets. Each set contains an increasing number of features from dataset * to guarantee that the mutation and crossover implementation do not result in an empty dataset. In my algorithm, the evaluation metric is measured based on r^2 score since it helps denote how close the data fit. Crossover and mutation ensure that new population with different sets of features are generated and evaluated properly. Since Genetic Algorithm is a non-greedy algorithm, it will find the optimal set of features. It is important that Genetic Algorithm runs on the same model as the training model for prediction.

Models

Since the dataset was complex, I decided to incorporate the data and extraction methods with scikit-learn's built in models. This prevents me from making computational mistakes while also helping me cover a wider range of models to see which one works best for the data I picked. I will be evaluating linear regression, polynomial regression, support vector regression, random forest regression, and k-nearest neighbors [6]. For each model, the data evaluated will be from the original data *, PCA's resulting data, and GA's resulting data. For the evaluation of the models, we will use the RMSE method to assess the prediction error and r^2 to measure the fitness of the regression models.

(i) Linear regression

```
Linear regression with original data
RMSE metric: 2.3702908077958242
r2 score: 0.8879818694166538
Linear regression with data resulted from PCA
RMSE metric: 0.6987270049502963
r2 score: 0.5200393756753787
Linear regression with data resulted from Genetic Algorithm
RMSE metric: 2.3681115415757876
r2 score: 0.888187755635678
```

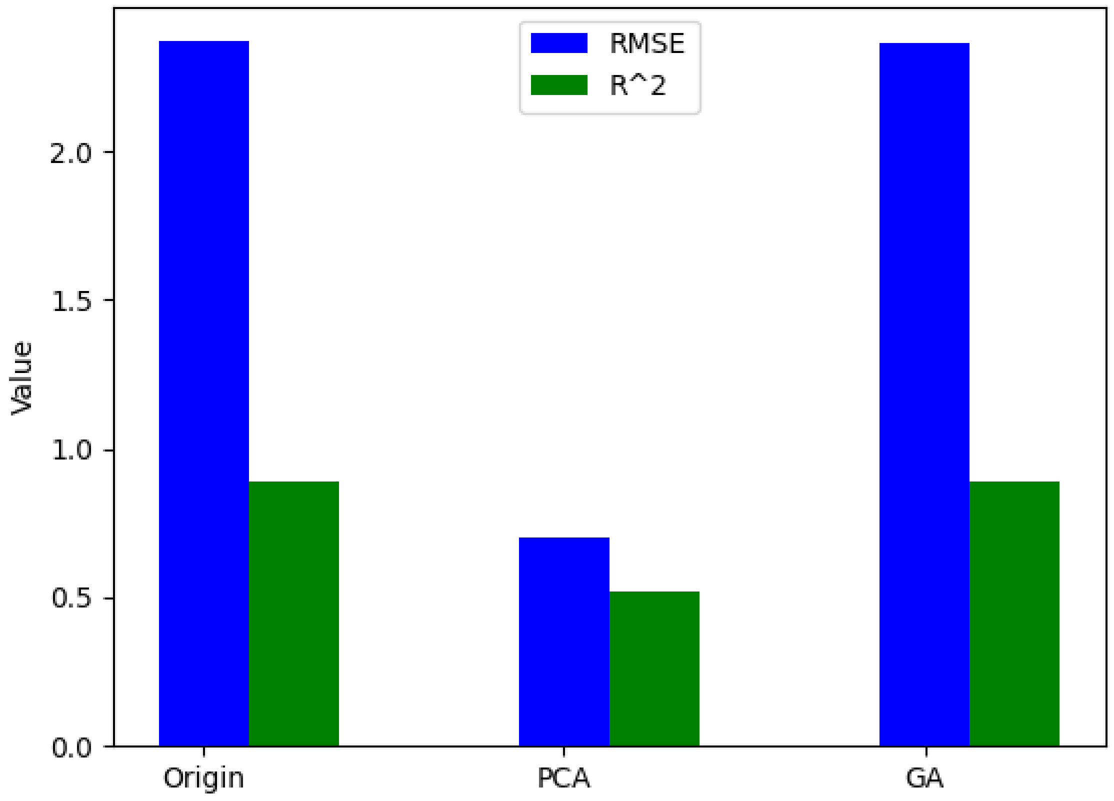


Fig2. Metrics evaluation comparison for linear regression

(ii) Polynomial regression

```
Polynomial regression (2nd degree) with original data
RMSE metric: 1.2676895478004266
r2 score: 0.9679586448915473
Polynomial regression (2nd degree) with data resulted from PCA
RMSE metric: 0.5145437690336214
r2 score: 0.7397233554043362
Polynomial regression (2nd degree) with data resulted from Genetic Algorithm
RMSE metric: 1.084884492062101
r2 score: 0.9765332979916295
```

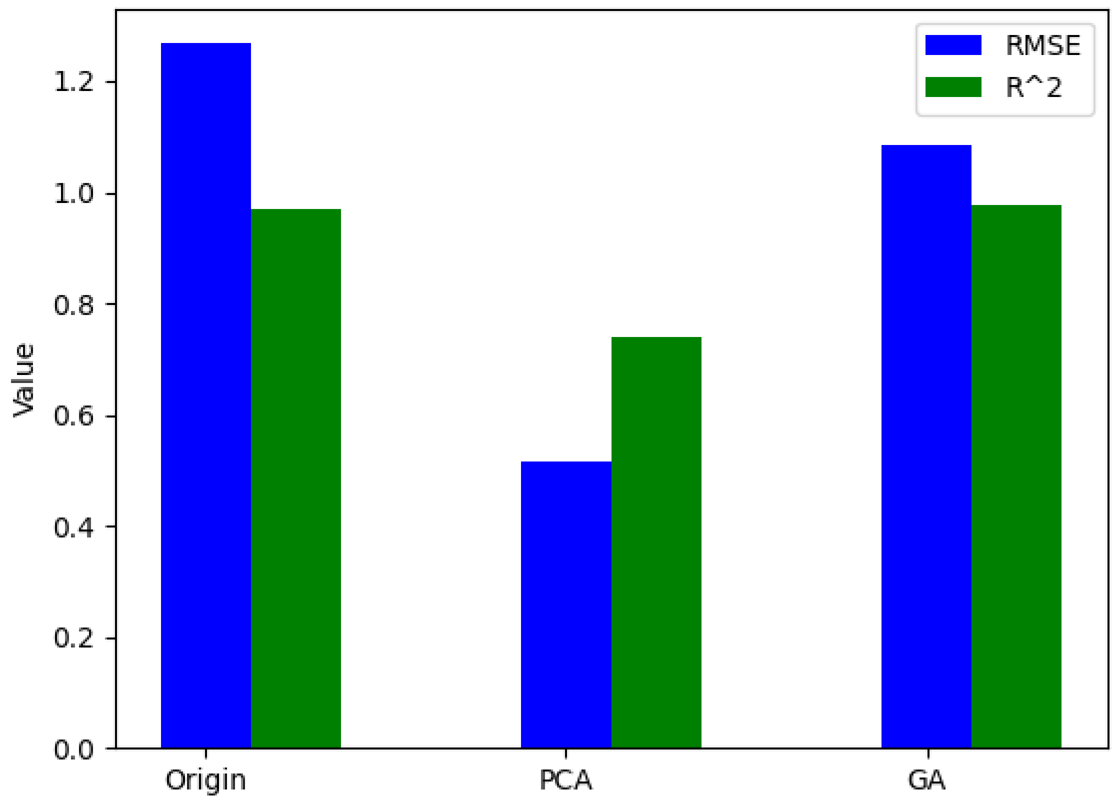


Fig3. Metrics evaluation comparison for polynomial regression (2nd degree)

(iii) Support vector regression

```
Support vector regression with original data
RMSE metric: 3.1638314739057956
r2 score: 0.8004224254774467
Support vector regression with data resulted from PCA
RMSE metric: 0.3331033161780562
r2 score: 0.8909191621503774
Support vector regression with data resulted from Genetic Algorithm
RMSE metric: 2.912878488589606
r2 score: 0.8308274900943451
```

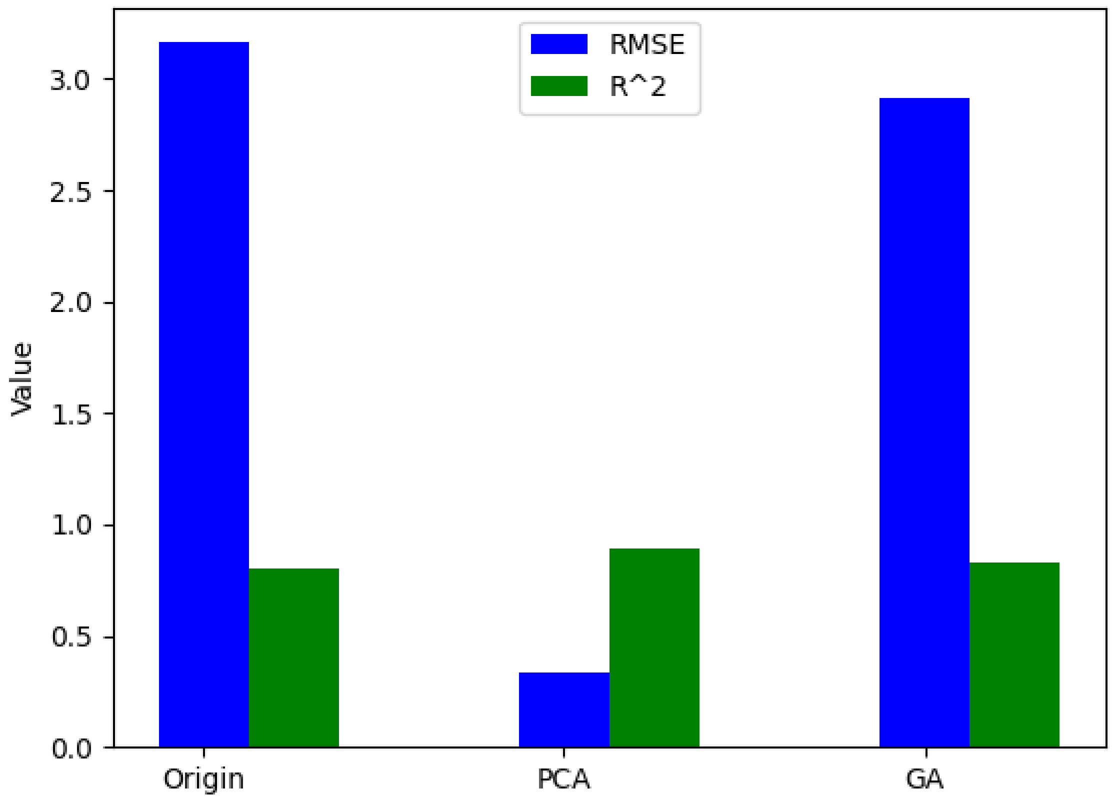


Fig4. Metrics evaluation comparison for support vector regression

(iv) Random forest regression

```
Random forest regression with original data
RMSE metric: 0.5308750027560505
r2 score: 0.9943808631571625
Random forest regression with data resulted from PCA
RMSE metric: 0.41211221196258924
r2 score: 0.8330365075397603
Random forest regression with data resulted from Genetic Algorithm
RMSE metric: 0.5145861831892968
r2 score: 0.9947203966197212
```

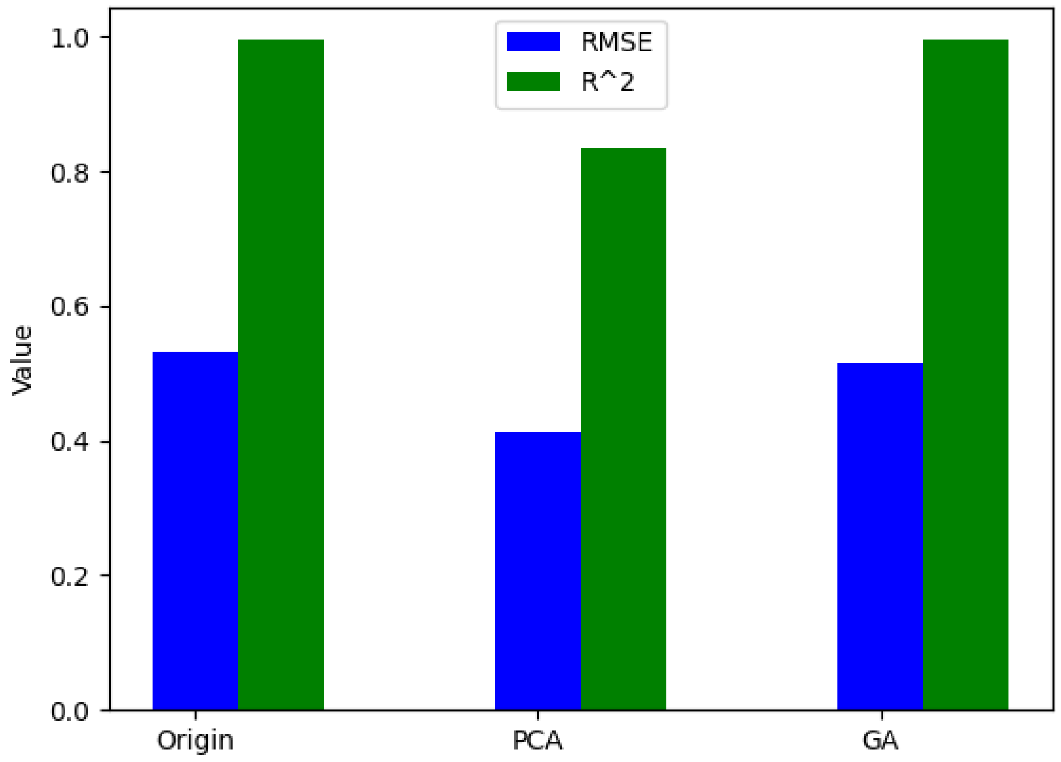


Fig5. Metrics evaluation comparison for random forest regression

(v) K-nearest neighbors

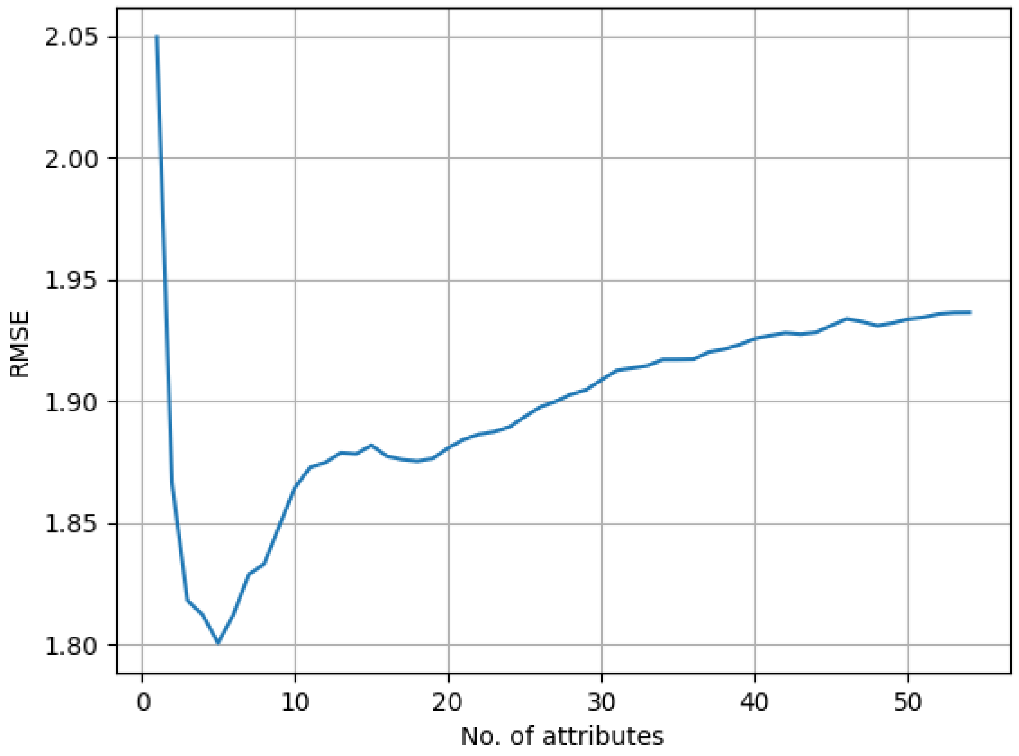


Fig6. RMSE vs. numbers of attributes for data *

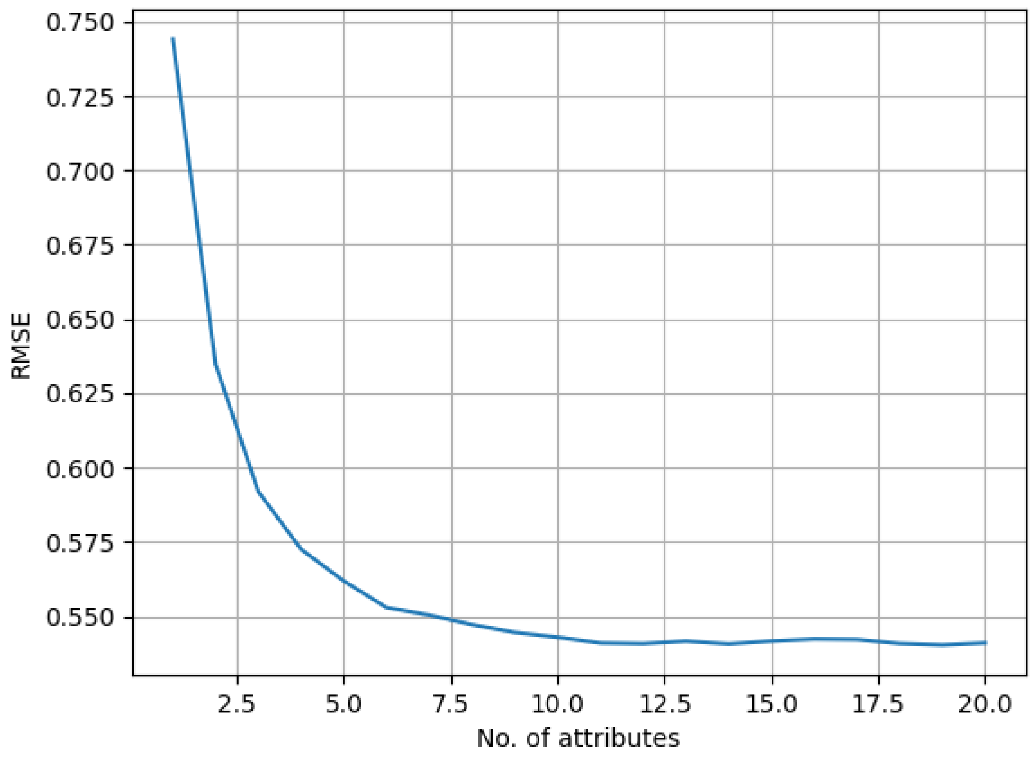


Fig7. RMSE vs. numbers of attributes for PCA's data

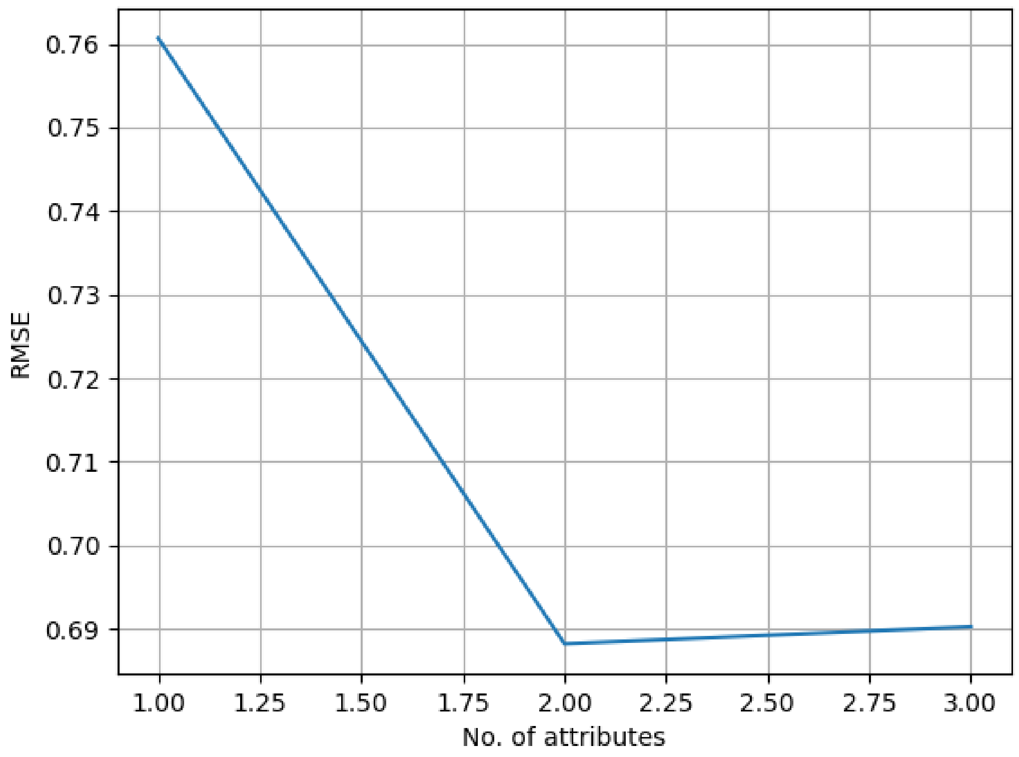


Fig8. RMSE vs. numbers of attributes for GA's data

Discussion

In general, the models chosen work well with the dataset. To my surprise, PCA performs generally better than GA in terms of root mean squared error. However, it does not work as well when considering coefficient of determination. A possible explanation would be the mean subtraction and standardization which helps scale the data well to avoid noise but in return it suffers from the loss of data correlation between features.

(i) It is to be expected that linear regression would have a hard time predicting this dataset because of how complex the relationship between features are. I am surprised that the Genetic Algorithm only performs slightly better than training and predicting with the original dataset. However, given that my GA implementation for linear regression still helps reduce the number of features by approximately 10, I would say that it helps improve my intuition of GA as an effective dimensionality reduction method. PCA's RMSE is significantly less since the PoV mentioned earlier helps choose a solid amount of transformed features that can reduce the noise productively.

(ii) It turns out that polynomial regression would also have a hard time predicting this dataset. However, in this case, we can see a better gap between the GA's attributes set and the original dataset. The GA implementation for polynomial regression of 2nd degree, though performing a bit worse than the linear regression, still reduces the number of features by about 7. PCA continues to deliver the best result.

(iii) Even though suppport vector regression is non-probabilistic and powerful with different kernel functions depending on our objective, the default parameters can sometimes overfit the data when we implement GA. Nonetheless, it usually helps narrow down more than half of the data *. It also helps me further identify two important attributes, value_eur and age, which are essential in the market place. This is because a player's value depends on how he has been performing so far, what title has he achieved, how fit he currently is, and so on.

As mentioned since SVR is prone to overfitting data, r^2 score may not be the best metric in this case. However, we can see a better gap between the original data * and GA's data, and this provides a solid evidence that GA is an effective method.

(iv) Even though there is only a slight difference between the original dataset and GA's dataset. GA actually helps remove about 20 features that have low weight in the dataset. With random forest considering various trees and then decide the best configuration, it helps the training and predicting step become more robust against noise in data. Looking at the results, PCA still performs the best, which helps us conclude that PCA is the best method to use for this dataset.

(v) For k-nearest neighbors, GA converges to contain only player's value, skill moves, and age. These are all important attributes of a player that any scout would want to look for. We can see that the original data has a lot of noise, but it seems that having about 5 attributes from the original data could reduce the RMSE significantly. Based on experimentation with other models, it would seem that age and player's value have a heavy weight in the dataset. For PCA, since we are scaling the data for both values and the target, our choice of 20 principle components remain a solid option since the RMSE decreases as we proceed and PCA yields less error than the other two approaches.

In general, there are many prominent attributes that correlate to a player's overall quality. In my program, I have seen a strong correlation between age, skill ratings, player's value, player's wage, and overall rating. This matches with my understanding that in most club transfer, scout agents would start out recruiting players based on those bases. An example would be Kylian Mbappé, a young player from Paris Saint-Germain F.C. He is currently 20 years old, and his value is 93.5 million euros (as of FIFA 20). This largely due to his recent success with the club as well as winning the World Cup 2018 with France. With these strong values, he is a potential player for any big football club to put him on their watchlist. His overall rating is 89, which is a significant improvement since he was only around 80 three to four years ago.

Conclusion

PCA is a great features extraction method that could help us engineer our data with affordable amount of computation and level of complication. It is important that each of the steps (mean subtraction, standardization, eigendecomposition of covariance matrix, etc.) is as equally essential. We want to transform the data onto the unit scale where mean is 0 and variance is 1 so that we could optimize the performance of our model. For eigendecomposition of covariance matrix, the eigenvalues and eigenvectors are the "heart" of PCA. The eigenvalues help explain the variance of data along the new feature axes determined by the eigenvectors. Therefore, PCA, and the program in general, would not be completed if we miss any of the steps.

Similarly, GA is also a great features extraction tool, but it can be a bit computational heavy depending on the models we are using. Each step and function in GA requires careful implementation, since there are many edge cases to account for such as empty dataset or duplicated attributes in a population's set. The crossover and mutation steps have more weights since they are essential in creating new population with better fitness than previous generations. It is shown in this project that PCA performs better than GA, much to my surprise.

Through the models I have tested, random forest regression is the most balanced when it comes to performance. Despite its heavy computational cost when using with GA, it does deliver good results that are better than those of the other models. Even though the dataset was complex, linear regression still managed to give reasonable results, but it would seem that the data follows a higher order as 2nd degree polynomial did outperform linear regression. Support vector regression delivers the smallest error with PCA, but maybe the default parameters do not work relatively well for GA or original data. K-nearest neighbors helps reinforce that PCA is the best method for this dataset although GA is still a strong candidate for dimensionality reduction and overall performance.

In conclusion, it is great to also know that my speculation about player scouting is correct based on which player's attributes having more influence than the other. Even though it would be a better option to retrieve players' information in real time, this project still provides me some important insights for PCA, GA, and regression models. For the future, the goal is to continue finding a stable website so that web scraping would be implemented. Furthermore, there are a few more models that I would like to test the data on, and I want to gear my focus towards hyperparameter tuning since this will help boost the performance of the models to obtain more optimized results.

References

[1] https://www.kaggle.com/stefanoleone992/fifa-20-complete-player-dataset?select=players_20.csv

[2] https://mml-book.github.io/book/mml-book.pdf

[3] http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf

[4] http://www.cs.ucc.ie/∼dgb/courses/tai/notes/handout12.pdf

[5] https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html

[6] https://scikit-learn.org/stable/index.html

[7] https://matplotlib.org

[8] https://numpy.org

[9] https://pandas.pydata.org

Appendix

The code can be found at https://github.com/TriPham2110/computational-ds. Any comments are welcome.