National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Murugappan Murugappan
Student ID: x19239831

School of Computing
National College of Ireland

Supervisor: Mr. Aaloka Anant

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Murugappan Murugappan		
Student ID:	X19239831		
Programme:	Data Analytics	**Year:**	2022
Module:	MSc Research Project		
Lecturer:	Mr. Aaloka Anant		
Submission Due Date:	31/01/2022		
Project Title:	Football Player Selection Based on Positions and Skills Using Ensemble Machine Learning and Similarity Measure Techniques		
Word Count:	**1915** **Page Count: 12**		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Murugappan Murugappan
Date:	31/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Murugappan Murugappan
x19239831

1 Introduction

This configuration manual document helps the end-users to reproduce the entire implementation of the project "**Football Player Selection Based on Positions and Skills Using Ensemble Machine Learning and Similarity Measure Techniques".** It comprises hardware and software requirements, the python program required for implementing this research approach starting from data pre-processing, EDA, feature selection, model building, hyperparameter tuning of the models, and the evaluation of the models. Also, the python program developed for creating the web app in finding similar players using cosine similarity is explained below.

2 Hardware Requirements

This research is implemented successfully on the "Lenovo IdeaPad S540-15IML D" laptop. To reproduce this research implementation, the system should have the basic configuration mentioned below or with advanced configurations

Operating System: Windows 10
Processor : i5-10210U CPU
RAM : 8 GB
Storage : 256 GB SSD

3 Software Requirements

This research code is implemented using jupyter notebook, which is an interactive web-based platform and with python programming. The code is delivered as .ipynb files for reproducing and the web app is also developed using the python inbuilt streamlit libraries. Below is the procedure to install the jupyter notebook in two ways

3.1 Download and Installation of Jupyter Notebook using ANACONDA

- Download ANACONDA from [1] and install anaconda simply by clicking next and finish the installation process. Then Install the jupyter notebook easily from the anaconda navigator by just simply clicking on Install.

[1] https://www.anaconda.com/products/individual

3.2 Installation of Jupyter Notebook using the commands

- Follow the instructions mentioned [2] to install the jupyter notebook using the commands.

3.3 Documentation for Jupyter Notebook

- Please refer to the [3] documentation for further clarity regarding the jupyter notebook and its installation.

4 Dataset

Once the setup is done by downloading and installing the jupyter notebook, Upload all the three .ipynb files in the current working directory. The FIFA 21 dataset can be downloaded from Kaggle [4] and placed under the same working directory where the .ipynb files are placed. When you are running the .ipynb files, the data can be properly read from the data file in the same working directory.

5 Importing Libraries

To implement this research right from cleaning the data, EDA, feature selection, model building, hyperparameter tuning, evaluation can be done with the help of certain python inbuilt libraries. To start with the implementation, let's import all the required libraries first as shown below in Figure 1.

Importing Libraries to build Ensemble Machine Learning Algorithms and Similarity Measures Techniques

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# For example, here's several helpful packages to load

import numpy as np                  # for performing linear algebraic operations
import pandas as pd                 # for performing data processing, CSV file I/O (e.g. pd.read_csv)
from   pandas import DataFrame      # importing dataframe from pandas to work with dataframe
from   scipy.spatial import distance # to find the euclidean distances between arrays
import warnings                     # to report the warning scenarios
from   sklearn.metrics import classification_report, confusion_matrix # to measure the quality of the model predictions
import matplotlib.pyplot as plt     # to work with plots
import seaborn as sns               # to work with statistical graphs
import plotly.express as px         # to work with matrix/ column oriented data
from sklearn.feature_selection import SelectKBest # Imported for feature selection technique
from sklearn.feature_selection import chi2        # Imported for feature selection technique
from sklearn.preprocessing import MinMaxScaler    # to convert the features to given range
from collections import Counter                   # to count the total number of values as dictionary
from imblearn.over_sampling import SMOTE,ADASYN   # used to distribute the data evenly over sampling technique
from sklearn.preprocessing import RobustScaler,StandardScaler,LabelEncoder,LabelBinarizer # imported for transforming ti
from sklearn.preprocessing import StandardScaler  # To perform standard scaling
```

Figure 1: Imported Libraries

[2] https://jupyter.org/install
[3] https://jupyter.org/documentation
[4] https://www.kaggle.com/umeshkumar017/fifa-21-player-and-formation-analysis

6 Data Pre-Processing

Once the data is downloaded and read into jupyter notebook, it is raw data and with many discrepancies. The model will not perform well with this raw data. So, the data has to be cleaned and processed before feeding it to the ensemble machine learning models. A few pre-processing steps are applied to clean the raw data like removing the unwanted columns, replacing the null values, feature engineering, label encoding is shown in below Figure 2.

Data Pre-Processing

```
features = ['Name', 'Age', 'Nationality', 'Overall', 'Potential', 'Club', 'Value', 'Wage', 'Special',
       'Preferred Foot', 'Weak Foot', 'Skill Moves','International Reputation', 'Work Rate', 'Body Type',
        'Position', 'Height', 'GK Reflexes']
# Incorporating those features in the dataframe
df = data[features]
df.shape
df['Volleys'].replace({np.NaN:data['Volleys'].mean()},inplace=True)
df['Curve'].replace({np.NaN:data['Curve'].mean()},inplace=True)
# function for transforming each positions to particular main position into 4 major positions as forward, midfielder, de
def complex_function(vc):
  if vc in ['ST','RW','RS','LW','CF','LS','LF','RF']:
    return 'Forward'
  elif vc in ['RM','LM','LCM','CM','CAM','RAM','RCM','LDM','CDM','RDM','LAM']:
    return 'Midfielder'
  elif vc in ['RB','CB','LB','RCB','RWB','LCB','LWB']:
    return 'Defender'
  else:
    return 'Goalkeeper'
# Applying function to create new field based on certain transformation
df_new['Grouped_Position'] = df_new['Position'].apply(complex_function)
# Label Encoding to convert the features to model understandable format
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in range(0,x.shape[1]):
    if x.dtypes[i]=='object':
        x[x.columns[i]] = le.fit_transform(x[x.columns[i]])

print(x)
```

Figure 2: Few Data Pre-Processing steps involved in this implementation

```
In [28]: # function for transforming each positions to particular 27 positions into their abbreviated positions
def complex_function(vc):
  if vc in ['RM']:
    return 'Right Midfielder'
  elif vc in ['ST']:
    return 'Striker'
  elif vc in ['LM']:
    return 'Left Midfielder'
  elif vc in ['RW']:
    return 'Right Wing'
  elif vc in ['LCM']:
    return 'Left Centre Midfielder'
  elif vc in ['RS']:
    return 'Right Safety'
  elif vc in ['RB']:
    return 'Running Back'
  elif vc in ['LW']:
    return 'Left Winger'
  elif vc in ['CM']:
    return 'Centre Midfielder'
  elif vc in ['CB']:
    return 'Cornerback'
  elif vc in ['CDM']:
    return 'Centre Defensive Midfielder'
  elif vc in ['CAM']:
    return 'Centre Attacking Midfielder'
  elif vc in ['LB']:
    return 'Linebacker'
  elif vc in ['RAM']:
    return 'Right Attacking Midfielder'
  elif vc in ['RCM']:
    return 'Right Centre Midfielder'
  elif vc in ['RCB']:
    return 'Right Centre Back'
  elif vc in ['RWB']:
    return 'Right Wing Back'
  elif vc in ['LDM']:
    return 'Left Defensive Midfielder'
  elif vc in ['LAM']:
    return 'Left Attacking Midfielder'
  elif vc in ['LCB']:
    return 'Left Centre Back'
  elif vc in ['CF']:
    return 'Cente Forward'
  elif vc in ['LS']:
    return 'Long Snapper'
  elif vc in ['GK']:
    return 'Goalkeeper'
  elif vc in ['LWB']:
    return 'Left Wing Back'
  elif vc in ['LF']:
    return 'Left Forward'
  elif vc in ['RDM']:
    return 'Right Defensive Midfielder'
  else:
    return 'Right Forward'
```

Figure 3: Transformation used for predicting 27 different positions

Figure 3, shows the transformation used to transform the position field and to feed it to the ensemble machine learning models to predict 27 different positions.

7 EDA

Exploratory Data Analysis

```
In [13]: nat_cnt=df.groupby('Nationality').apply(lambda x:x['Name'].count()).reset_index(name='Counts')
         nat_cnt.sort_values(by='Counts',ascending=False,inplace=True)
         top_20_nat_cnt=nat_cnt[:20]
         fig=px.bar(top_20_nat_cnt,x='Nationality',y='Counts',color='Counts',title='Nationwise Representation in the FIFA Game')
         fig.show()

In [14]: cnt_best_avg=df.groupby('Nationality').apply(lambda x:np.average(x['Overall'])).reset_index(name='Overall Ratings')
         cnt_best_cnt=df.groupby('Nationality').apply(lambda x:x['Overall'].count()).reset_index(name='Player Counts')
         snt_best_avg_cnt=pd.merge(cnt_best_avg,cnt_best_cnt,how='inner',left_on='Nationality',right_on='Nationality')
         sel_best_avg_cnt=snt_best_avg_cnt[snt_best_avg_cnt['Player Counts']>=200]
         sel_best_avg_cnt.sort_values(by=['Overall Ratings','Player Counts'],ascending=[False,False])
         px.scatter(sel_best_avg_cnt,x='Overall Ratings',y='Player Counts',color='Player Counts',size='Overall Ratings',hover_dat

In [18]: pos_cnt=df.groupby('Position').apply(lambda x:x['Name'].count()).reset_index(name='Counts')
         pos_cnt.sort_values(by='Counts',ascending=False,inplace=True)
         top_20_pos_cnt=pos_cnt[:20]
         fig=px.bar(top_20_pos_cnt,x='Position',y='Counts',color='Counts',title='Positionwise Player counts in FIFA 21')
         fig.show()

         # position abbreviation for reference

         #   'Right Midfielder', 'Striker', 'Left Midfielder', 'Right Wing',
         #         'Left Centre Midfielder', 'Right Safety', 'Running Back',
         #         'Left Winger', 'Centre Midfielder', 'Cornerback',
         #         'Centre Defensive Midfielder', 'Centre Attacking Midfielder',
         #         'Linebacker', 'Right Attacking Midfielder',
         #         'Right Centre Midfielder', 'Right Centre Back', 'Right Wing Back',
         #         'Left Defensive Midfielder', 'Left Attacking Midfielder',
         #         'Left Centre Back', 'Cente Forward', 'Long Snapper', 'Goalkeeper',
         #         'Left Wing Back', 'Left Forward', 'Right Defensive Midfielder',
         #         'Right Forward'

In [21]: cond_1=df['Overall']!=df['Potential']
         cond_2=df['Age']<25
         df_fil=df[cond_1 & cond_2]
         potential_play=df_fil[['Name','Age','Nationality','Club','Potential','Position','Overall','Value']]
         potential_play.sort_values(by='Potential',ascending=False,inplace=True)
         top_potential_play=potential_play[:50]
         fig=px.scatter(potential_play,x='Age',y='Potential',size='Potential',color='Age',hover_data=['Name','Age','Nationality',
         fig.show()
```

Figure 4: Code Developed for EDA

Once the data is cleaned, Exploratory Data Analysis EDA can be done on the data to understand the data better. The EDA gives more insights about the particular data and based on that conclusions can be drawn concerning the data and it can help in taking decisions in the further steps of implementation. The few EDA codes developed are shown in Figure 4 and EDA visualization in this research can be shown below in Figure 5, Figure 6, Figure 7, Figure 8.

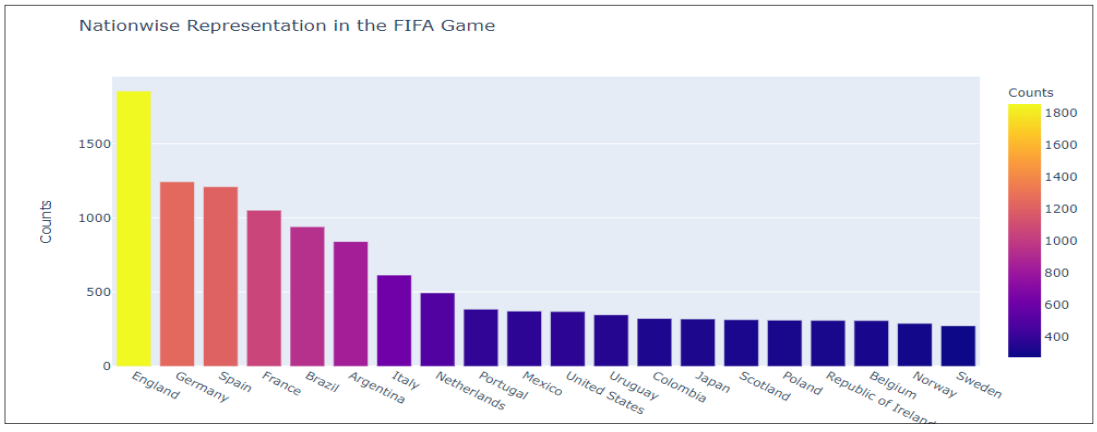

Figure 5: Nation wise representation in the FIFA Game

From Figure 5, it is visible that the countries such as England, Germany, Spain, France, and Brazil produce the highest number of players in the FIFA game while the country such as Ireland, Belgium, Norway, Sweden produces the lowest number of players in the FIFA game.

The highest number of players is from England alone where around 1800 players are representing the FIFA game.

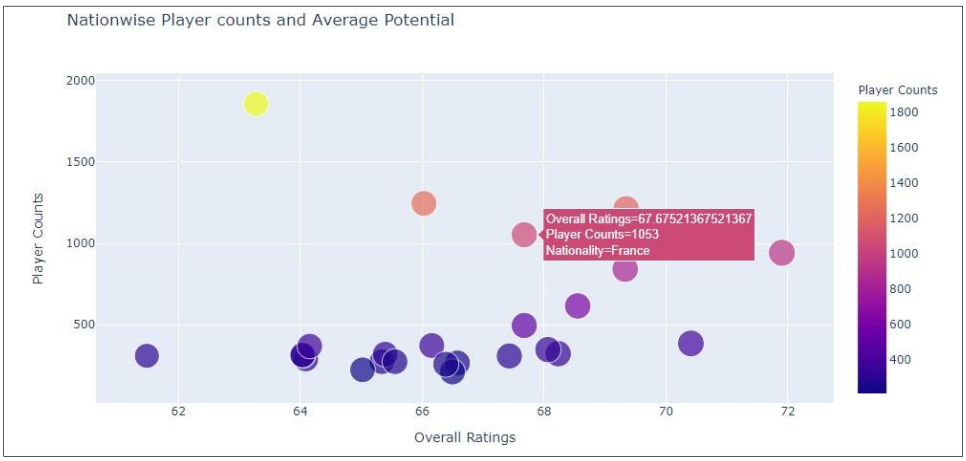

Figure 6: Nation wise player and average potential

From Figure 6, The average potential is determined from the overall ratings and the count of the players in particular countries. From the graph, we can see that the yellow dot represents the country England where the overall rating is less but the count of the player is more and the last point represent the country Brazil where the overall rating is more but the player count is less. So we can say that the average potential of the country Brazil is more than any other country despite the less Player count.

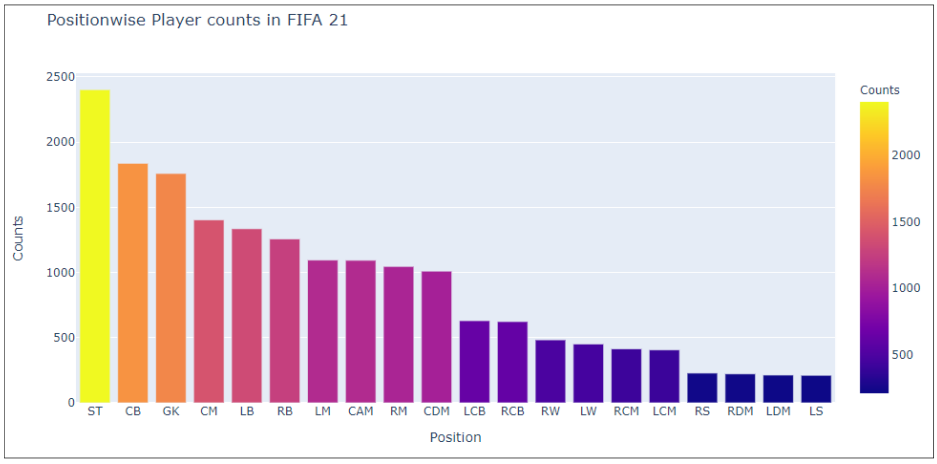

Figure 7: Position Wise Player

From Figure 7, ST represents the striker position, CB represents the center-back position and GK represents the goalkeeper position. So we can say that these three positions have the highest number of players in FIFA 2021 games.

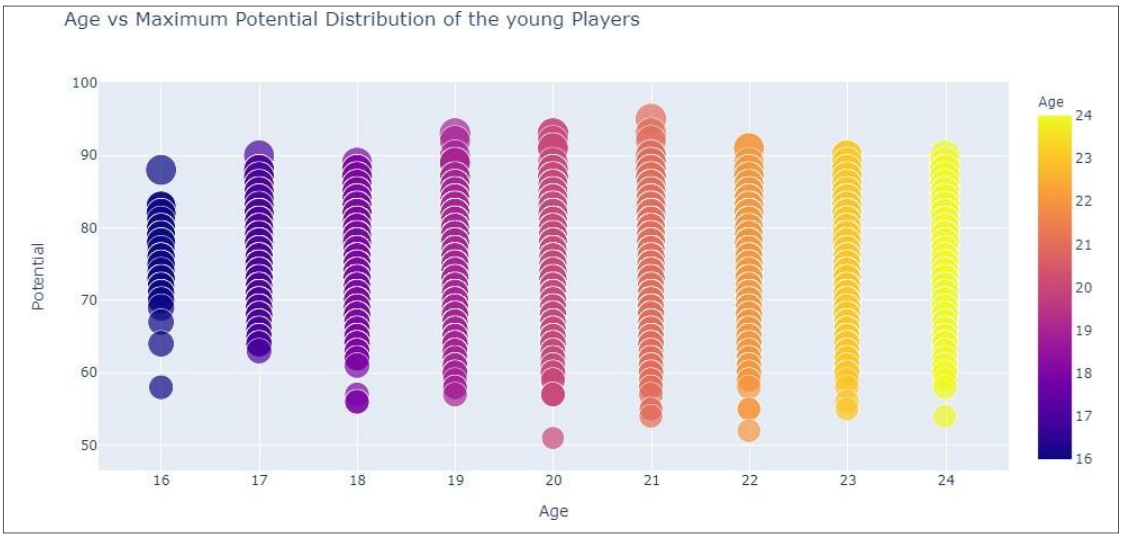

Figure 8: Age Vs Maximum Potential Distribution

Figure 8, tells about the age of a player related to the maximum potential of that particular player. From the graph, we can see that the player who is 20 or 21 years old has the highest number of potential as there are many players in this age range that have the maximum potential.

8 Feature Selection

Once the EDA is completed, it gives a clear understanding of the data and helps to draw some conclusions like which fields are important in building the ensemble models and so those fields can be retained and the rest can be removed from the data. This can be done using feature selection techniques. The SelectKBest technique is used in this research to select the best features that contribute to the model prediction and the code for the feature selection is shown in below Figure 9.

Feature selection using selectkBest technique

```
In [21]: # converted the features to given range and selected the top 30 features
         x_norm = MinMaxScaler().fit_transform(x)
         chi_selector = SelectKBest(chi2, k=num_feats)
         chi_selector.fit(x_norm, y)
         chi_support = chi_selector.get_support()
         chi_feature = x.loc[:,chi_support].columns.tolist()
         print(str(len(chi_feature)), 'selected features')

         30 selected features
```

Figure 9: Code for SelectKBest Feature Selection Technique

9 Model Building

Once the features are selected using the feature selection techniques, those features are fed to the ensemble machine learning models to learn and predict the player positions. Model Building using 4 different Ensemble Machine Learning Models like Support Vector Classifier, Logistic Regressor Classifier, Random Forest Classifier, and Decision Tree Classifier which works on bagging and boosting approach and will help in improving the model performance when compared to all the other traditional models. This model building consists of two approaches one for predicting position with 4 major positions and another

approach for predicting 27 different positions. Let's see the implementation of each model one by one as follows.

9.1 Implementation of Support Vector Classifier Base and Tuned Models (Major positions and 27 different positions)

```
In [41]: from sklearn.svm import SVC
         from sklearn.model_selection import RandomizedSearchCV
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         svc = SVC( kernel= 'rbf', gamma= 0.1, C= 9)
         svc.fit(X_train, y_train)
         y_pred = svc.predict(X_test)
         # print('Training Accuracy: %1.3f.' % svc.score(X_train, y_train))
         #10-fold cross validation score
         cv = cross_val_score(estimator = svc, X = X_train, y = y_train, cv =3)
         SVC_cv = cv.mean()
         print("10 fold cross validation :", SVC_cv)
         from sklearn.metrics import accuracy_score
         SVC_ac = accuracy_score(y_pred,y_test)
         print("accuracy :", SVC_ac)
         from sklearn.metrics import precision_score
         SVC_p = precision_score(y_pred,y_test,average='weighted')
         print("precision :", SVC_p)
         from sklearn.metrics import recall_score
         SVC_r = recall_score(y_pred,y_test,average='weighted')
         print("recall :", SVC_r)
         from sklearn.metrics import f1_score
         SVC_f1 = f1_score(y_test, y_pred,average='weighted')
         print("F1 Score :", SVC_f1)
```

Figure 10: Code Implemented for SVC

The support vector classifier for both base and tuned models in predicting the 4 major positions and in predicting the 27 different positions is achieved by using the code snippet shown in the Figure 10 with few differences in data transformation.

9.2 Implementation of Logistic Regression Classifier Base and Tuned Models (Major positions and 27 different positions)

```
In [46]: from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         lr = LogisticRegression(penalty = 'l2', C = 1 )
         lr.fit(X_train, y_train)
         y_pred = lr.predict(X_test)
         #10-fold cross validation score
         cv = cross_val_score(estimator = lr, X = X_train, y = y_train, cv =3)
         log_cv = cv.mean()
         print("10 fold cross validation :", log_cv)
         from sklearn.metrics import accuracy_score
         log_ac = accuracy_score(y_pred,y_test)
         print("accuracy :", log_ac)
         from sklearn.metrics import precision_score
         log_p = precision_score(y_pred,y_test,average='weighted')
         print("precision :", log_p)
         from sklearn.metrics import recall_score
         log_r = recall_score(y_pred,y_test,average='weighted')
         print("recall :", log_r)
         from sklearn.metrics import f1_score
         log_f1 = f1_score(y_test, y_pred,average='weighted')
         print("F1 Score :", log_f1)
```

Figure 11: Code Implemented for LRC

The Logistic Regression classifier for both base and tuned models in predicting the 4 major positions and in predicting the 27 different positions is achieved by using the code snippet shown in the Figure 11 with few differences in data transformation.

9.3 Implementation of Random Forest Classifier Base and Tuned Models (Major positions and 27 different positions)

```
In [52]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         rf = RandomForestClassifier(n_estimators = 1860, min_samples_split = 2, min_samples_leaf = 4, max_features='auto', max_
         rf.fit(X_train, y_train)
         # Predicting the Test set results
         y_pred = rf.predict(X_test)
         #10-fold cross validation score
         cv = cross_val_score(estimator = rf, X = X_train, y = y_train, cv =3)
         rf_cv = cv.mean()
         print("10 fold cross validation :", rf_cv)
         from sklearn.metrics import accuracy_score
         rf_ac = accuracy_score(y_pred,y_test)
         print("accuracy :", rf_ac)
         from sklearn.metrics import precision_score
         rf_p = precision_score(y_pred,y_test,average='weighted')
         print("precision :", rf_p)
         from sklearn.metrics import recall_score
         rf_r = recall_score(y_pred,y_test,average='weighted')
         print("recall :", rf_r)
         from sklearn.metrics import f1_score
         rf_f1 = f1_score(y_test, y_pred,average='weighted')
         print("F1 Score :", rf_f1)
```

Figure 12: Code Implemented for RFC

The Random Forest classifier for both base and tuned models in predicting the 4 major positions and in predicting the 27 different positions is achieved by using the code snippet shown in Figure 12 with few differences in data transformation.

9.4 Implementation of Decision Tree Classifier Base and Tuned Models (Major positions and 27 different positions)

```
In [60]: # Fitting Decision Tree Classification to the Training set
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         classifier = DecisionTreeClassifier(min_samples_leaf = 6, max_features=7, max_depth= None, criterion = 'gini')
         classifier.fit(X_train, y_train)

         # Predicting the Test set results
         y_pred = classifier.predict(X_test)

         #10-fold cross validation score
         cv = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv =3)
         dc_cv = cv.mean()
         print("10 fold cross validation :", dc_cv)
         from sklearn.metrics import accuracy_score
         dc_ac = accuracy_score(y_pred,y_test)
         print("accuracy :", dc_ac)
         from sklearn.metrics import precision_score
         dc_p = precision_score(y_pred,y_test,average='weighted')
         print("precision :", dc_p)
         from sklearn.metrics import recall_score
         dc_r = recall_score(y_pred,y_test,average='weighted')
         print("recall :", dc_r)
         from sklearn.metrics import f1_score
         dc_f1 = f1_score(y_test, y_pred,average='weighted')
         print("F1 Score :", dc_f1)
```

Figure 13: Code Implemented for DTC

The Decision Tree classifier for both base and tuned models in predicting the 4 major positions and in predicting the 27 different positions is achieved by using the code snippet shown in the Figure 13 with few differences in data transformation.

10 Hyperparameter Tuning of Ensemble Models

```
# SVC parameters
svc_params = {'C': range(1, 10, 1), 'gamma': np.arange(0.1, 1, 0.1), 'kernel': ['rbf', 'linear']}

# LRC parameters
lr_param={"C":np.logspace(-3,3,7), "penalty":["l1","l2"],}# l1 lasso l2 ridge
param_comb = 100

# RFC parameters
from sklearn.model_selection import RandomizedSearchCV
max_features = ['auto', 'sqrt']
n_estimators = range(200, 2000, 10)
max_depth = range(10, 110, 10)
min_samples_split = range(2, 10, 1)
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
param_comb = 100

# DTC parameters
param_dist = {"max_depth": [3, None],
              "max_features": range(1, 9),
              "min_samples_leaf": range(1, 9),
              "criterion": ["gini", "entropy"]}

random_search = RandomizedSearchCV(estimator = svc, param_distributions = svc_params, n_iter = 20, cv = 5, verbose=2, random_
random_search.fit(X_train, y_train)

print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best hyperparameters:')
print(random_search.best_params_)
results = pd.DataFrame(random_search.cv_results_)
```

Figure 14: Code Implemented for Hyperparameter Tuning of Models

The hyperparameter tuning of the models can be implemented using the code snippet shown in Figure 14 by changing the parameter values of each model found using hyperparameter optimization techniques.

11 Similarity Measures Techniques

Similarity Measures Using Cosine Similarity and Euclidean Distance

```
# Calculating cosine similarity after standard scaling
cossim=[]
for i in range (0,len(Similarity1)):
    cossim.append(1 - distance.cosine(Similarity1[p_ind],Similarity1[i]))
pd.Series(cossim)
sim2={"name":sn,"cossim":cossim}
sim2=pd.DataFrame(sim2)

# Sorting in ascending based on cosine similarity values after standard scaling
sim2.iloc[sim2["cossim"].sort_values(ascending=False).index].head(10)

# calculating similarity based on euclidean distance after standard scaling and sort in ascending values
dist=[]
for i in range (0,len(Similarity1)):
    dist.append(distance.euclidean(Similarity1[p_ind],Similarity1[i]))
pd.Series(dist)
sim={"name":sn,"distance":dist}
sim=pd.DataFrame(sim)
sim.iloc[sim["distance"].sort_values().index].head(10)
```

Figure 15: Code Implemented for Similarity Measures Techniques

The code snippet is shown in Figure 15 is used to develop the similarity measures approaches to find the similar players using cosine similarity and Euclidean distance approach.

12 Evaluation Results of the Above Models

The developed models have to be evaluated before moving to the market. So, that the quality of the product can be tested and can confirm the use of humans. The importance of evaluating the product after development is explained in detail by (Ma and Ladisch, 2019).

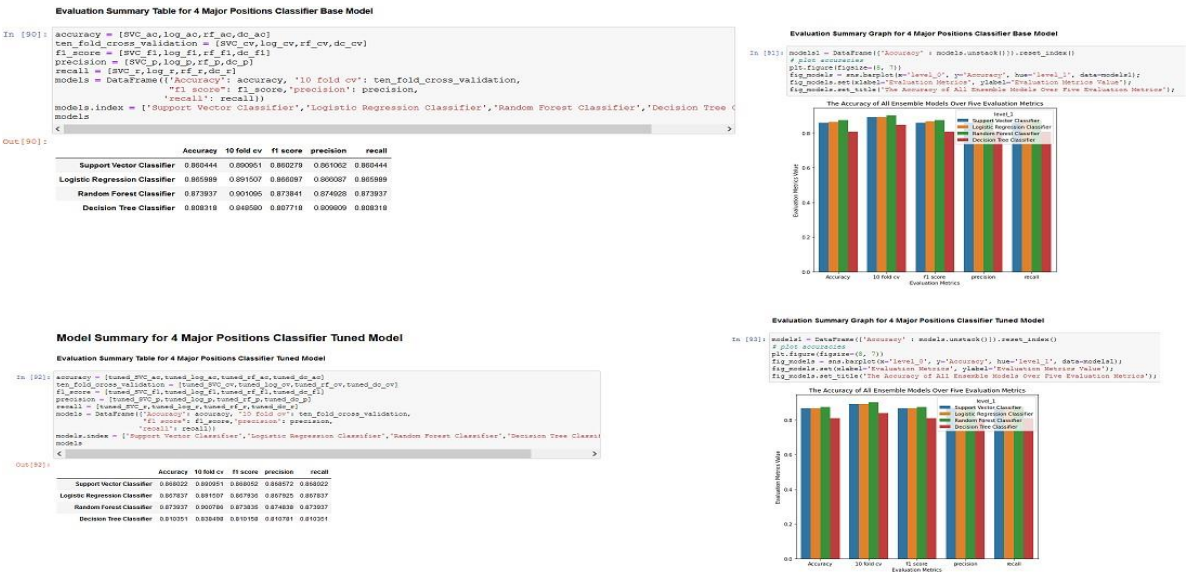

Figure 16: Evaluation Results of Base and Tuned Models predicting 4 Major Positions

Random Forest Classifier performs better with above Both Base and Tuned models evaluation metrics when compared to all other 3 models in predicting the major 4 positions of the player is inferred from the Figure 16.

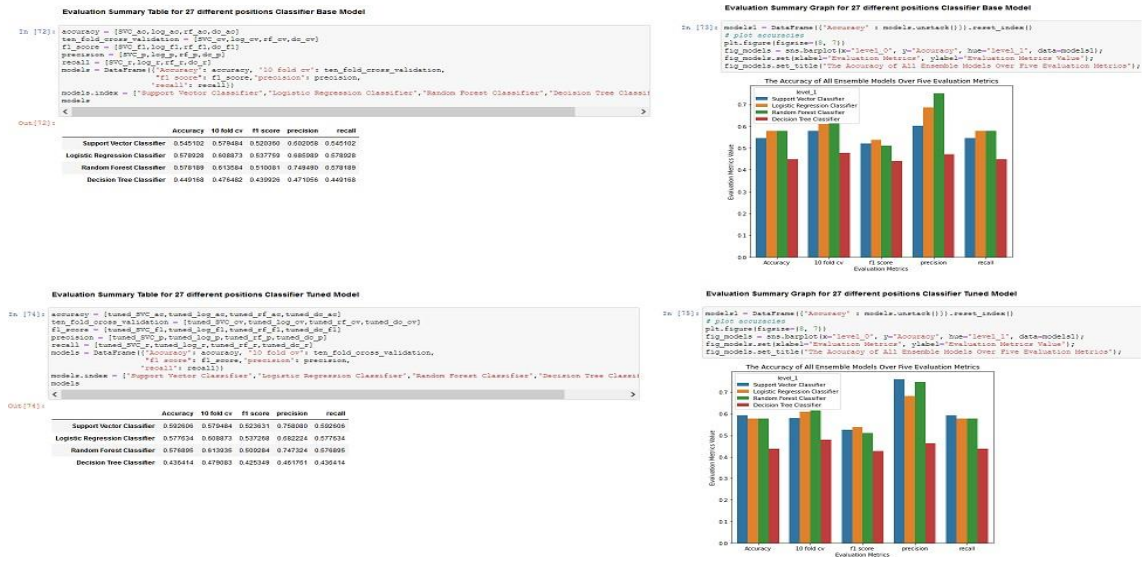

Figure 17: Evaluation Results of Base and Tuned Models predicting 27 Different Positions

Overall, the conclusion for the 27 different positions classifier models comparing Base and Tuned models is that Tuned Support Vector Classifier performs better when compared to all other Classifier models evaluation metrics in predicting the 27 different positions of the player is inferred from Figure 17.

Figure 18: Evaluation Results of Similarity Measures Techniques

Cosine similarity performs better when compared to the euclidean distance approach and helps to find the more similar player and in replacing the particular player is inferred from Figure 18.

13 Creating and Running the Web App

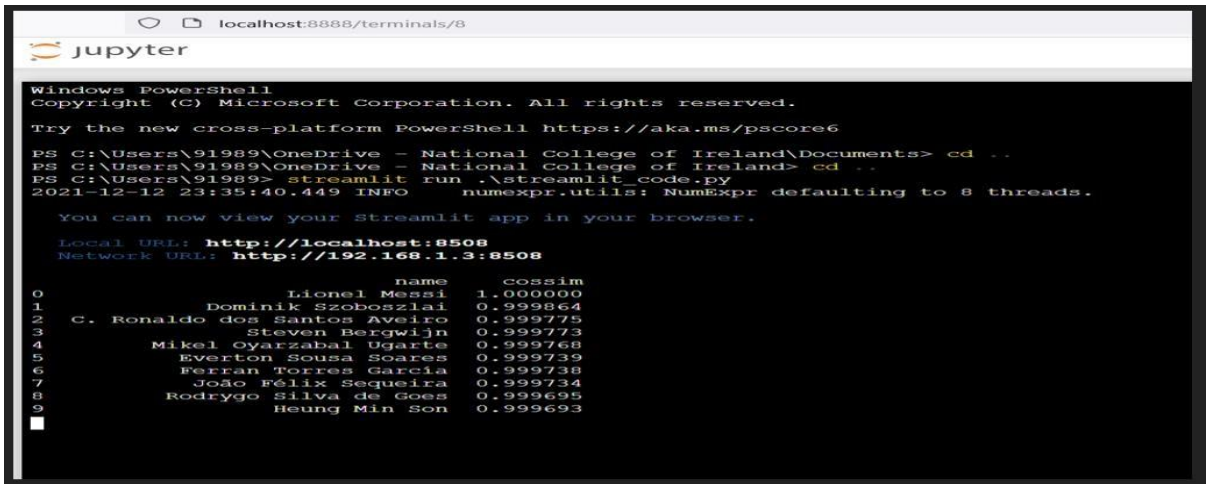

Figure 19: Running in Terminal to get the Web App

Finally, a web app is created for the cosine similarity approach in finding similar players for replacement based on skills and other attributes. This web app is developed using a stream lit python inbuild library and created a custom function to pass the required attributes from the similarity data frame. Once the user enters the particular player name and by that time custom function developed uses the required attributes from the similarity.csv file and displays the top 10 similar players for the value passed. The Similarity.csv file is just in which the data is transferred after pre-processing to the CSV file from the main similarity measure code. Figure 19 shows how to start up the web app on the local server by running the particular streamlit_code.py file. While running, make sure that all the files like fifa21.csv,

similarity.csv, and streamlit_code.py files are in the same current working directory along with all the three .ipynb files.

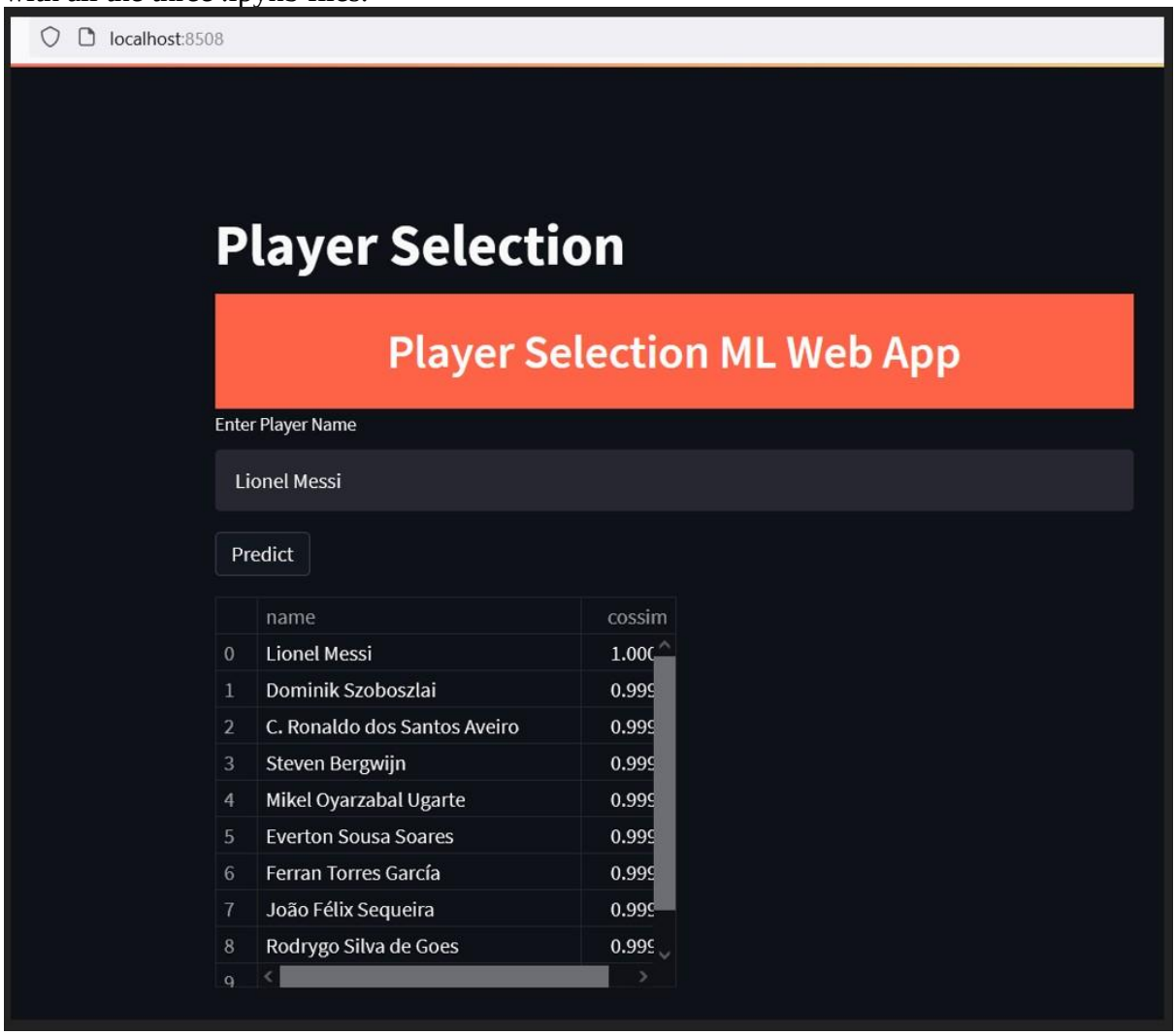

Figure 20: Web App Created Using Stream lit python Inbuilt Library

Once after running the streamlit_code.py file to start up the Player Selection ML web app, the web app appears as shown in Figure 20. The user can enter the player name as required and click predict and it will generate the top 10 similar players based on the skills and other attributes of that player. So, those similar players can be replaced in the team in place of that particular player.

References

Ma, L. and Ladisch, M. (2019). Evaluation complacency or evaluation inertia? A study of evaluative metrics and research practices in Irish universities. *Research Evaluation*.