Predicting high-potential FIFA players using individual performance data

Merete Lutz, Jake Barnabe, Simon Frew, Waleed Mahmood

DSCI 522, Group 17

Summary

We attempt to construct a classification model using an RBF SVM classifier algorithm which uses FIFA22 player attribute ratings to classify players' potential with target classes "Low", "Medium", "Good", and "Great". The classes are split on the quartiles of the distribution of the FIFA22 potential ratings. Our model performed reasonably well on the test data with an accuracy score of 0.84, with hyperparamters C: 100 & Gamma: 0.01. However, we believe there is still significant room for improvement before the model is ready to be utilized by soccer clubs and coaching staffs to predict the potential of players on the field instead of on the screen.

Introduction

One of the most challenging jobs for sports coaches is deciding which players will make a positive addition to the team (US National Soccer Players, 2023). A key step in evaluating which players to add to a team is predicting how their skill level will change over time. We can think of this in terms of their potential. FIFA22 by EA sports is the world's leading soccer video game. For each year's release, they rate players' skill levels in various aspects of the game such as shooting, passing, defending, etc. and give each player an overall rating as well as a rating of each player's potential.

Here we ask if we can use a machine learning model to classify players by their potential given their attribute ratings. Answering this question is important as developing a model that can accurately predict the potential of players on FIFA22 could then be applied to the evaluation of soccer players in real life and be employed by coaches and scouts to help soccer clubs make good decisions on which players to add to the team and which to let go.

In [1]:
```
import os
import requests
import warnings
import zipfile
```

```python
import numpy as np
import pandas as pd
import altair as alt

from hashlib import sha1
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_validate,
    train_test_split,
)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB, GaussianNB
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import loguniform, randint, uniform
from sklearn.model_selection import RandomizedSearchCV

warnings.filterwarnings('ignore', category=FutureWarning)
alt.data_transformers.enable("vegafusion");
pd.set_option('display.max_rows', 200)
```

Methods

Data

The data used in this analysis are from the video game FIFA22 by EA Sports. The data were downloaded with authentication from Kaggle and without authentication from Sports-Statistics.com. Within documentation, these were were scraped from a publicly available website (https://sofifa.com/) with a permissive `robots.txt`.

Each row of the dataset corresponds to a single player, and contains biometric information, ratings for various skills, like shooting accuracy, passing, dribbling, and player wages and market value.

Analysis

The Radial Basis Function (RBF) Support Vector Machine (SVM) RBF SVM model was used to build a classification model to predict whether a player has high potential or not (found in the potential column of the data set). The variables included in our model were selected from the list of different player statistics that are part of the dataset, including the statistics on their `speed`, `dribbling`, `shooting` etc. These are the variables that were used as features to fit the model. The hyperparameters `gamma` and `C` were chosen through the use of the automated optimization method from `scikit-learn` called `RandomizedSearchCV`. The Python programming language (Van Rossum and Drake 2009) was used and the following Python packages were used to perform the analysis: Numpy (Harris et al. 2020), Pandas (McKinney 2010), altair (VanderPlas et al., 2018), SkLearn (Pedregosa et al. 2011) and SciPy (Pauli Virtanen, et al., 2020). The code used to perform the analysis and create this report can be found here: https://github.com/UBC-MDS/2023_dsci522-group17/.

Results and Discussion

To look at whether each of the predictors might be useful to predict the class of the target variable `potential`, we plotted the distributions of each predictor from the training data set and coloured the distribution by class (Low: blue, medium: orange, Good: red, Great: green). These distributions are drawn up after we have scaled all of the features in the training dataset. In doing this, we can see that most of the distributions, for the features we have filtered to keep, have overlap but their spreads and centers are different, except for `height_cm` and `weight_kg` - which overlap almost completely across the different classes of `potential` - and `value_eur` - which has no distribution for classes other than `great` for `potential` - so we omit them from our model.

Through our model selection process, we were able to determine that the best model in our case would be an RBF SVM model. To determine the values for the hyperparameters that would give us the best estimator, we used the hyperparameter optimization method `RandomizedSearchCV` to perform a 5-fold cross validation, so that we are able to get the most suitable hyperparamters to obtain the best possible model to estimate and predict the class for `potential`.

We observe that the optimal hyperparameter values are `C: 100` and `Gamma: 0.01`. The training accuracy obtained with these `hyperparameters` is 0.786. And the accuracy of our model is 0.84, i.e. it predicted quite well when run on the test data.

In [2]:
```
# download dataset
# method adapted from: https://github.com/ttimbers/breast_cancer_predictor_p
url = "https://sports-statistics.com/database/fifa/fifa_2022_datasets.zip"

request = requests.get(url)

with open(os.path.join("data", "fifa_2022_datasets.zip"), "wb") as f:
    f.write(request.content)

with zipfile.ZipFile(os.path.join("data", "fifa_2022_datasets.zip"), "r") as
```

```
    zip_file.extract("players_22.csv", path="data")

os.remove(os.path.join("data", "fifa_2022_datasets.zip"))

# data selection
df_raw = pd.read_csv("data/players_22.csv", encoding="utf-8", low_memory=Fal
df_raw
```

Out[2]:

	sofifa_id	player_url	short_name	
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Li Mes
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Le
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristia
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neyr S
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevir
...	...	...	...	
19234	261962	https://sofifa.com/player/261962/defu-song/220002	Song Defu	
19235	262040	https://sofifa.com/player/262040/caoimhin-port...	C. Porter	Caoi
19236	262760	https://sofifa.com/player/262760/nathan-logue/...	N. Logue	Na C
19237	262820	https://sofifa.com/player/262820/luke-rudden/2...	L. Rudden	L
19238	264540	https://sofifa.com/player/264540/emanuel-lalch...	E. Lalchhanchhuaha	Lalchl

19239 rows × 110 columns

In [3]:

```
# Selecting columns for analysis
df_processed = df_raw.loc[:, ["potential",
                              "value_eur",
                              "wage_eur",
                              "age",
                              "height_cm",
                              "weight_kg",
                              "weak_foot",
                              "skill_moves",
                              "pace",
                              "shooting",
                              "passing",
                              "dribbling",
                              "defending",
                              "physic",]]
```

```
# Dropping observations with missing values
df_processed = df_processed.dropna()
```

In [4]:
```
# Binning the target class 'potential' into 4 categories
df_processed['potential'] = pd.cut(x=df_processed['potential'], bins=[0, 67,
                     labels=['Low', 'Medium', 'Good', 'Great'])
df_processed
```

Out[4]:

	potential	value_eur	wage_eur	age	height_cm	weight_kg	weak_fo
0	Great	78000000.0	320000.0	34	170	72	
1	Great	119500000.0	270000.0	32	185	81	
2	Great	45000000.0	270000.0	36	187	83	
3	Great	129000000.0	270000.0	29	175	68	
4	Great	125500000.0	350000.0	30	181	70	
...	...	...	...	...	...	...	
19234	Low	70000.0	1000.0	22	180	64	
19235	Low	110000.0	500.0	19	175	70	
19236	Low	100000.0	500.0	21	178	72	
19237	Low	110000.0	500.0	19	173	66	
19238	Low	110000.0	500.0	19	167	61	

17041 rows × 14 columns

In [5]:
```
# Create the split
fifa_train, fifa_test = train_test_split(df_processed, test_size=0.3, random

fifa_train.to_csv("data/processed/fifa_train.csv")
fifa_test.to_csv("data/processed/fifa_test.csv")
```

In [6]:
```
# Pre-processing
passthrough_feats = ["potential"]
numeric_feats = ['value_eur', 'wage_eur', 'age', 'height_cm', 'weight_kg',
       'weak_foot', 'skill_moves', 'pace', 'shooting', 'passing', 'dribbling
       'defending', 'physic']

fifa_preprocessor = make_column_transformer(
    ("passthrough", passthrough_feats),
    (StandardScaler(), numeric_feats),
)

fifa_preprocessor.fit(fifa_train)
scaled_fifa_train = fifa_preprocessor.transform(fifa_train)
scaled_fifa_test = fifa_preprocessor.transform(fifa_test)

column_names = (passthrough_feats + numeric_feats)
scaled_fifa_train = pd.DataFrame(scaled_fifa_train, columns=column_names)
```

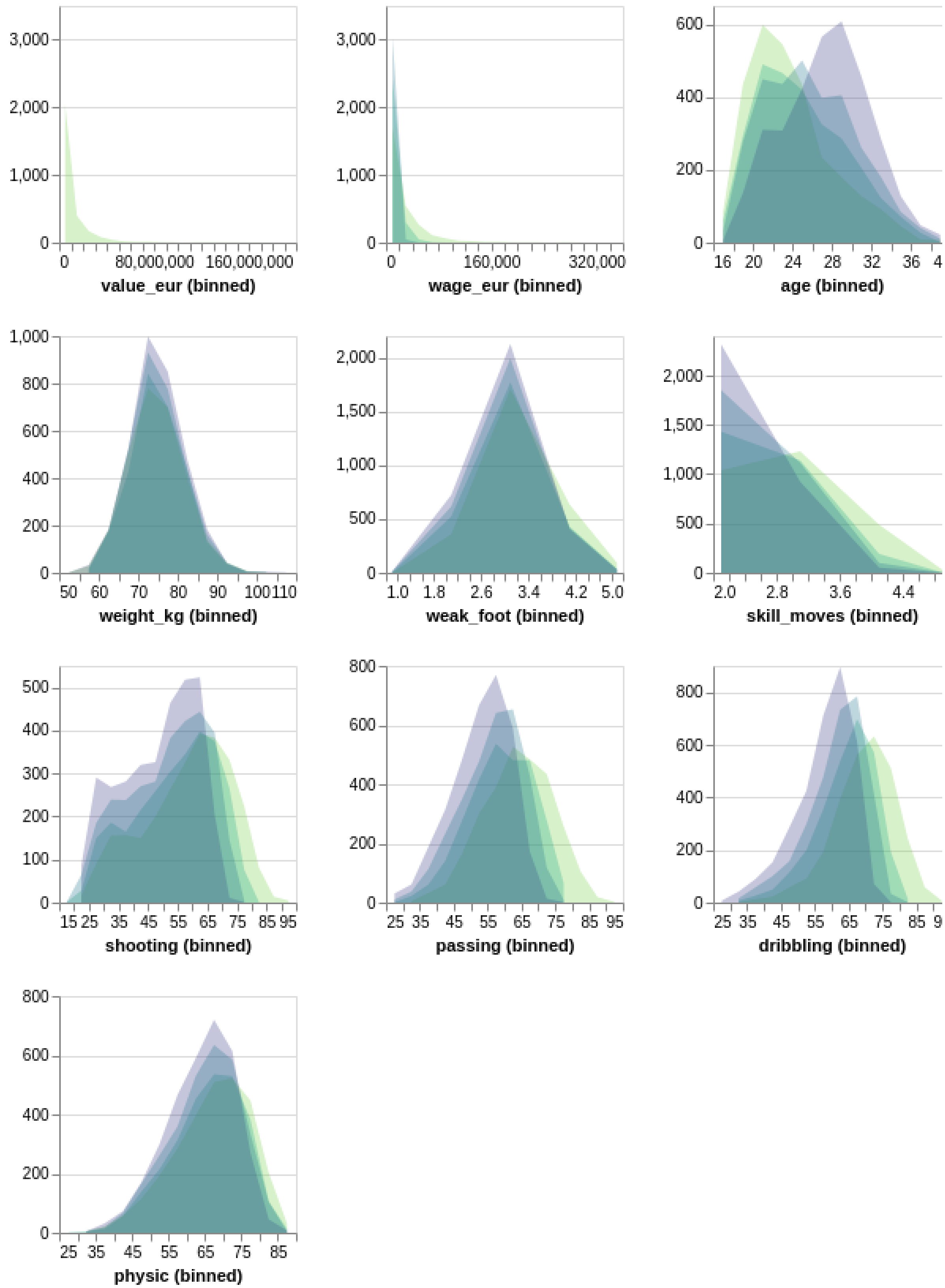
```python
scaled_fifa_test = pd.DataFrame(scaled_fifa_train, columns=column_names)

# Saving scaled data
scaled_fifa_train.to_csv("data/processed/scaled_fifa_train.csv")
scaled_fifa_test.to_csv("data/processed/scaled_fifa_test.csv")
```

In [7]:
```python
# Exploratory data analysis
stack_order = ['Low', 'Medium', 'Good', 'Great']

alt.Chart(fifa_train).mark_area(opacity=0.3).encode(
    alt.X(alt.repeat()).type('quantitative').bin(maxbins=20),
    alt.Y('count()', stack=None).title(''),
    alt.Color('potential').sort(stack_order).title("Potential").scale(scheme
).properties(
    width = 150,
    height = 150
).repeat(
    numeric_feats,
    columns = 4
)
```

Out[7]:



In [8]:

```
X_train = scaled_fifa_train.drop(columns = ['potential'])
y_train = scaled_fifa_train['potential']
X_test = scaled_fifa_test.drop(columns = ['potential'])
y_test = scaled_fifa_test['potential']
```

In [9]:
```
# Looking for the best model for our data
models = {
    "dummy": DummyClassifier(random_state=123),
    "Decision Tree": DecisionTreeClassifier(random_state=123),
    "KNN": KNeighborsClassifier(),
    "RBF SVM": SVC(random_state=123),
    "Naive Bayes": GaussianNB(),
    "Logistic Regression": LogisticRegression(max_iter=2000, multi_class="ov
}
```

In [10]:
```
# Function attributed to Leture 2 of 571 found at the below link
# https://pages.github.ubc.ca/MDS-2023-24/DSCI_571_sup-learn-1_students/lect
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation
    """
    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [11]:
```
results = pd.DataFrame()
for name, model in models.items():
    results[name] = mean_std_cross_val_scores(model, X_train, y_train, retur
results
```

Out[11]:

	dummy	Decision Tree	KNN	RBF SVM	Naive Bayes	Logistic Regression
fit_time	0.005 (+/- 0.000)	0.057 (+/- 0.002)	0.023 (+/- 0.002)	2.401 (+/- 0.018)	0.014 (+/- 0.001)	0.199 (+/- 0.006)
score_time	0.002 (+/- 0.000)	0.004 (+/- 0.001)	0.287 (+/- 0.008)	0.776 (+/- 0.005)	0.003 (+/- 0.000)	0.010 (+/- 0.000)
test_score	0.276 (+/- 0.000)	0.821 (+/- 0.006)	0.567 (+/- 0.005)	0.751 (+/- 0.009)	0.570 (+/- 0.007)	0.705 (+/- 0.005)
train_score	0.276 (+/- 0.000)	1.000 (+/- 0.000)	0.728 (+/- 0.004)	0.786 (+/- 0.002)	0.570 (+/- 0.002)	0.709 (+/- 0.002)

In [12]:
```
# Hyperparamater optimization for SVC model
param_dist = {
    "svc__C": [0.001, 0.01, 0.1, 1, 10, 100],
    "svc__gamma": [0.001, 0.01, 0.1, 1, 10, 100]
}
```

```python
pipe = make_pipeline(SVC(random_state=123))

random_search = RandomizedSearchCV(pipe,
                           param_dist,
                           n_iter=36,
                           n_jobs=-1,
                           return_train_score=True,
                           random_state=123)


random_search.fit(X_train, y_train)

pd.DataFrame(random_search.cv_results_)[
    [
        "mean_test_score",
        "mean_train_score",
        "param_svc__C",
        "param_svc__gamma",
        "mean_fit_time",
        "rank_test_score",
    ]
].set_index("rank_test_score").sort_index().iloc[:5]
```

Out[12]:

	mean_test_score	mean_train_score	param_svc__C	param_svc
rank_test_score				
1	0.811703	0.836310	100	
2	0.779678	0.948357	100	
3	0.778169	0.876069	10	
4	0.772132	0.785106	10	
5	0.752095	0.759411	100	

In [13]:
```python
pd.DataFrame(results["RBF SVM"])
```

Out[13]:

	RBF SVM
fit_time	2.401 (+/- 0.018)
score_time	0.776 (+/- 0.005)
test_score	0.751 (+/- 0.009)
train_score	0.786 (+/- 0.002)

In [14]:
```python
best_model = random_search.best_estimator_
best_model
```

Out[14]:

```
▸ Pipeline
  ▸ SVC
```

In [15]:

```
best_model.score(X_test, y_test)
```

Out[15]:

```
0.8374413145539906
```

Further Improvements

To improve our model further in the future, with the hopes of better predicting the potential of a player, there are a few improvements that can be made. First of all, we could include the growth of a player over the years, based on their performance in games. This would somewhat lead to us having a time-series dataset which we can use to create a feature that captures the growth of a player over the years. Second, we would include the effort that the player puts into their training. This can be a beneficial improvement that could lead to better predictive power in our model. Finally, we could include the reporting of the probability estimates of the prediction of the classes in for the `potential` of a player, so that a player scout knows with how much certainty a player might be classified as a `Great` player (for example).

References

US National Soccer Players. (2023). (rep.). How to evaluate soccer players. Retrieved from https://ussoccerplayers.com/soccer-training-tips/evaluating-players.

Harris, C.R. et al., 2020. Array programming with NumPy. Nature, 585, pp.357–362.

McKinney, Wes. 2010. "Data Structures for Statistical Computing in Python." In Proceedings of the 9th Python in Science Conference, edited by Stéfan van der Walt and Jarrod Millman, 51–56.

Pauli Virtanen, et al., and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), pp.2825–2830.

VanderPlas et al., (2018). Altair: Interactive Statistical Visualizations for Python. Journal of Open Source Software, 3(32), 1057, https://doi.org/10.21105/joss.01057

Van Rossum, Guido, and Fred L. Drake. 2009. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.