



UNIVERSITÀ DI PISA

MSc in Artificial Intelligence and Data Engineering

Soccer Ratings

Data mining and machine learning's
project documentation

Giuseppe Martino

Github repository: <https://github.com/giuseppemartino26/S-Ratings>

Table of contents

Introduction.....	2
The application	2
Requirements	2
Actors.....	2
Functional requirements	2
Non-functional requirements	3
UML Use case diagram	3
Structure and functioning of the application.....	3
Frameworks and technologies	3
Machine Learning.....	4
Dataset	4
Pre processing	4
Discretization of class attribute	4
Data trasformation	5
Conversion of categorical data.....	6
Dealing with missing values	6
Removing irrelevant attributes	7
Removing duplicates and inconsistencies.....	7
Normalization.....	7
Classification.....	8
Ordinal Classifier	9
Feature selection	11
Performances results.....	13
Statistical Significance	15
Final considerations.....	17
Application User Guide.....	17
References.....	22
Appendix.....	23
Hyperparameter selection	23
List of attributes selected in the chosen model.....	23

Introduction

Newspapers and sports magazines are used to evaluate the performance of individual players after a football match, but this is not a trivial task because it is difficult to arrive at an objective evaluation.

Indeed, usually happens that different raters assign different ratings to the same player since sometimes different criteria may be used to decide whether a footballer has played well or not and there is consistency with regard to the player's performance only in the case of players involved in crucial events in a match such as a decisive goal or assist (positive) or a missed penalty (negative). Anyway football is a team game and the success of a team involves many players and so the evaluation cannot be based just on events like goals and assists that by the way are few in a football match.

Soccer Ratings is an application designed to help with this task: it is intended to help a football coach and his staff in evaluating a players's performance in a match, assigning him a rating from 1 to 5, after entering various statistics about that player for that specific match.

This rating is then used to calculate the best starting line-up for the next match, for which the most fit players will be taken into account, i.e. those who have played the best in the last 5 matches.

The application

Requirements

Actors

- Not registered user
- User: the standard user of the application. A classic user can be a football coach who wants to have suggestions on the best formation to deploy

Functional requirements

A User can:

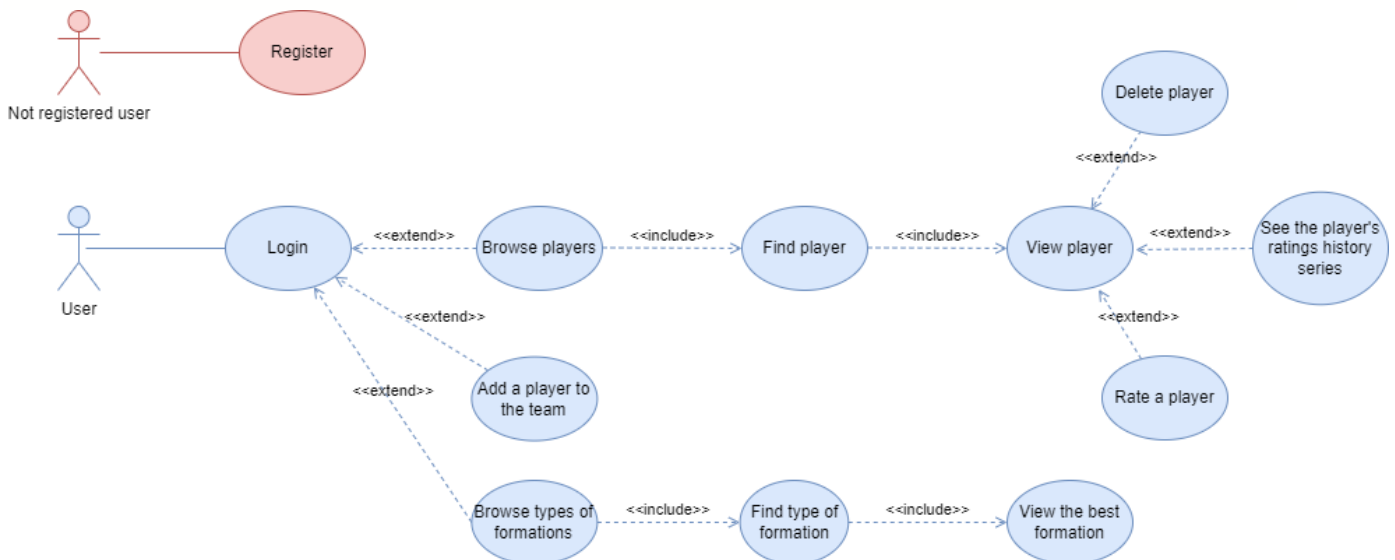
- Login
- Create his own team and modify it, adding new player or removing others
- Rate a player entering statistical information about a player's performance in a match
- See the best formation for the next match suggested by the application
- See the graph of a player's ratings history series

A not registered user can register to the application.

Non-functional requirements

- The application must be user-friendly and easy to use through a clear user interface.
- The application must provide fast responses to users requests
- The application must assign performance ratings using a machine learning model that has an acceptable prediction error

UML Use case diagram



Structure and functioning of the application

The application consists of a simple GUI through which users can interact with the application.

Data about users, players and ratings computed by the application are managed and stored in a database and the application interacts with the database in order to reply to users' requests.

When the user clicks on the "Rate a player" button of the GUI in order to rate the selected player, the application loads the machine learning model. Now the user can enter all the attributes required through the GUI and so the model can predict the rating which will be stored along with other information (like the date of the match) in the database.

The best formation is computed in this way: for each position are taken the players with the highest average rating of the last 5 matches.

Information about ratings are kept in the database for one year and after that will be deleted automatically.

Frameworks and technologies

All the machine learning stages (data pre-processing, feature selection, model learning and validation) have been implemented in a *Jupyter Notebook (analysis.ipynb)* in Python,

mainly using the *scikit-learn* library. Also *python-weka-wrapper3* APIs have been used in order to exploit some functionality from Weka.

The application was implemented in Python using *Pycharm* as IDE.

For the GUI the python library *TKinter* has been used.

The database used is a relational database (MySQL).

The project is available at the following GitHub repository:

<https://github.com/giuseppemartino26/S-Ratings>

Machine Learning

Dataset

The dataset (source: <https://www.kaggle.com/datasets/gabrielmanfredi/football-players-ratings>) is composed by 50652 instances and 63 attributes. It contains data about 789 different matches across 4 different competitions between 2016 and 2018: each record of the dataset is composed by the description of the performance of one player in one specific match through almost 60 different statistical data and the corresponding rating assigned to that player for that single match.

Ratings are taken from 6 different sport magazines and specialized websites: Kicker, Bild, SkySports, TheGuardian, WhoScored and SofaScore, so there can be different ratings assigned to the same player performance for a single match.

Pre processing

Discretization of class attribute

In the dataset ratings are taken from different raters and each rater uses their own scale:

- Kicker and Bild: 1 to 6 scale with best performances ordered with a descending order (1 is the best performance)
- All the other raters use a 1 to 10 scale with best performances ordered with an ascending sorting (10 is the best performance)
- SofaScore and WhoScored use algorithms to assign their ratings, so their ratings are continuous values

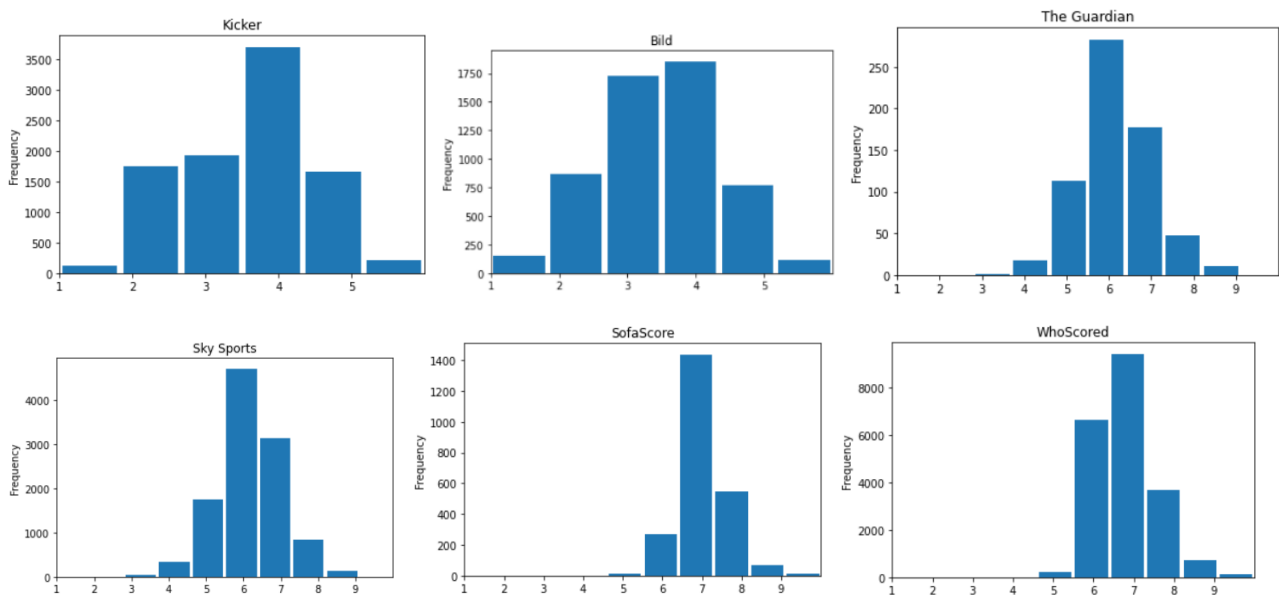
The objective is to transform the ratings in order to have a unique scale.

As we can see at the image below, even raters who use a scale from 1 to 10, do not actually use all numbers from 1 to 10, but the ratings are mostly limited in a range from 4 to 8 or 9. For this reason I decided to transform all the ratings in a 1 to 5 scale, where 1 is a very bad performance and 5 is an excellent performance.

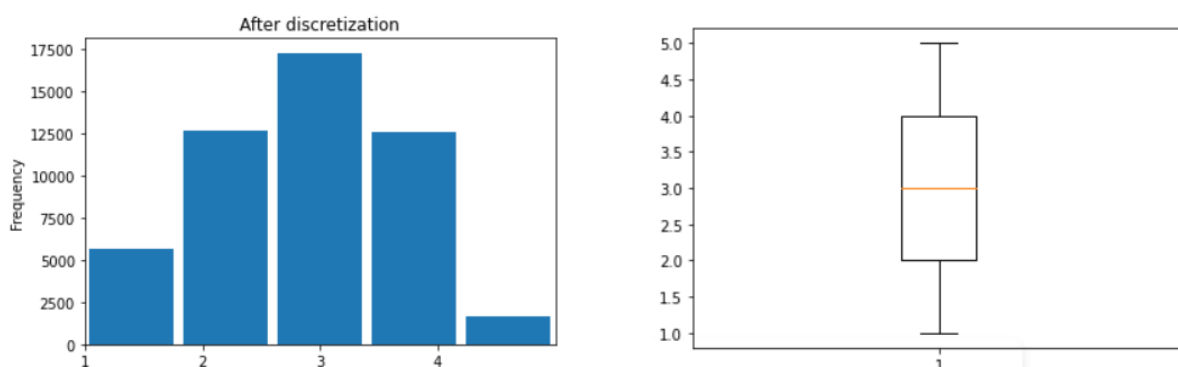
To achieve this goal, for each rater a discretization has been applied to the class attribute rating into 5 bins, merging then together the trasformed data, creating in this way our class composed by 5 ordinal values, using a clustering approach. In particular has been used the sklearn function *KBinsDiscretizer* with the parameter strategy 'kmeans', so values in each bin have the same nearest center of a 1D k means cluster.

Below we can see the class distribution for each rater and the class distribution of the dataset after the discretization.

In this way the objective is to classify the performance in one of these 5 categories.



After the discretization:



Data trasformation

There are two attributes in the dataset used to express the position of a player in the field: **pos** and **pos_role**, where the first one is the general role (Goalkeeper, Defender, Midfielder and Foreward) and the second one is the specific position covered in the field by the player.

The attribute `pos_role` is a categorical attribute with 16 different possible values and since during a match the coach can change strategies and tactics several times and the position of a player on the pitch can also change during the match, so it would be reductive in my opinion to assign a single position to a player in a match, while certainly the general role of a player is not subject to changes (a defender for example can change position several times in a match but will always be a defender throughout his career).

At the same time the attribute `'pos'` is also too much general, especially for midfielders: indeed there are more offensive midfielders, from whom goals, assists or general participation in the offensive phases are expected, unlike midfielders who play immediately in front of the defence.

For this reason I decided to use the information in `'pos_role'` to modify the values of the midfielders in `'pos'`: each midfielder is divided into `'Attacking Midfielder'`, `'Midfielder'` (central) and `'Defensive Midfielder'`. Then I decided to **drop the attribute `'pos_role'`**.

Conversion of categorical data

So the attribute `'pos'` is a categorical data.

Since many machine learning algorithms cannot operate with them but require all variables to be numeric, this means that the attribute `'pos'` had to be converted into a numerical form.

For this purpose the ***One-hot encoding*** was used. The result of this was to have six additional binary attributes, one for each possible role category.

Dealing with missing values

For many records, the `pos` attribute takes the value of `'Sub'`, which does not correspond to a role but indicates the fact that the player has taken over and not started the match from the first minute.

It can therefore be deduced that we have no information about the player's position when he took over as the game was in progress, so it has been treated as a missing value.

Before handling this missing value anyway, I used this information to create an additional binary attribute called **`'starter'`**, which indicates whether or not a player started in that match from the first minute.

Concerning the records for which the `'pos'` attribute was missing, were filled with the **mode** calculated on the group of records for that specific football player in that specific team.

However, there were still remaining instances of players who had always taken over and for whom the mode could not be calculated. Some of these were filled by hand, searching on the web their usual position, while others were deleted.

Removing irrelevant attributes

The following attributes have been then removed since were considered irrelevant for the classification purpose:

- competition
- date
- match
- rater
- team
- is_human

Removing duplicates and inconsistencies

After the removal of these attributes I had to deal with **duplicates** and **inconsistencies**.

Due to the fact that the dataset is an integration of performance and ratings, which come from different magazines, at this point of the pre-processing phase can be present:

- several instances of the same performance which may differ only in the rating - the class to be predicted: in the case where several magazines disagreed and therefore assigned different ratings for the same performance
- duplicate records in the case of agreement between several magazines.

The decision was to remove the duplicates.

Even records that differed only in class were also removed, in order to have greater accuracy (if for example we had three completely identical instances in the test set, only one could be predicted correctly, but we would have an error in the remaining two) and make the data more consistent and coherent.

Normalization

When dealing with algorithms that need feature scaling, like KNN, SVM or Logistic Regression, the **Z-score normalization** has been applied in order to normalize the data, using the scikit-learn function *StandardScaler*.

At the end of the pre processing phase, the dataset is composed by 6676 instances with 60 attributes.

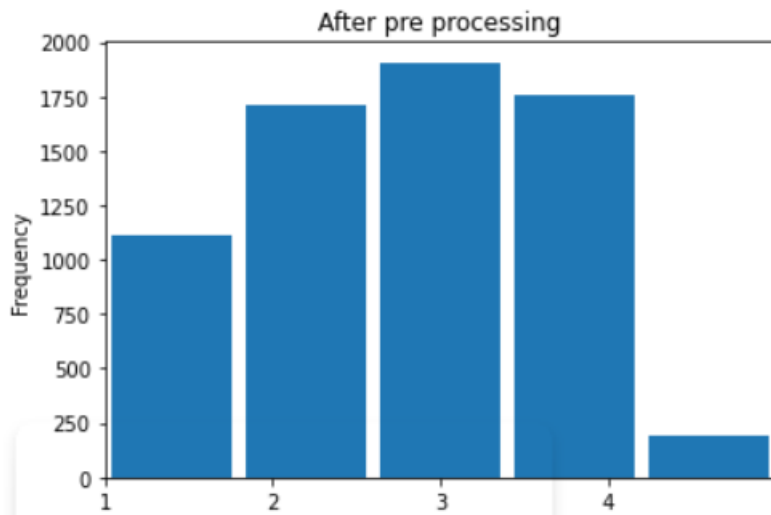


Figure 1 Class distribution after the pre processing

As we can see from the image above, the classes are **imbalanced**.

Classification

The problem of predict a rating from 1 to 5 can be considered a problem of **ordinal classification**. «Ordinal classification is a form of multi-class classification where there is an inherent ordering between the classes, but not a meaningful numeric difference between them» (Gaudette 2009) , «which shares some characteristics of multi-class classification and regression, however, in contrast to the former the order between class labels cannot be neglected, and, in contrast to the latter, the scale of the decision attribute is not cardinal» (K. Dembczyński 2007).

Since there is an order in the class, not all the misclassifications should be considered equal: miss-classify a “1” as “2” should not have the same impact as miss-classify a “1” as “5”.

Indeed in general for ordinal classification problems good metrics are MAE and RMSE (Gaudette 2009). Anyway in my analysis I had to consider also the imbalance of the dataset.

Considering this, the metric on which I focused the most in order to evaluate the performance of the models is the **Macro Averaged Mean Absolute Error** using the correspondent method present in the python library *imbalanced learn*. It computes each MAE for each class and average them, giving an equal weight to each class.

Indeed, as suggested in (S. Baccianella 2009) , the macro averaged version of the classic MAE and RMSE are better to use in order to cope with imbalanced datasets in ordinal regression problems even if they are equivalent to the classic MAE and RMSE when the datasets are perfectly balanced.

All the classifiers have been evaluated with a **10-fold cross validation**.

Since the dataset is imbalanced, performances of classifiers were tested also after having applied some rebalancing techniques as **SMOTE** and **RandomOverSampler**.

The tested classifiers are:

- KNN
- Support Vector Machine
- XGBClassifier
- Random Forest
- Logistic Regression
- OrdinalClassifier

For each classifier was performed the **hyperparameters tuning** using the **GridSearchCV** algorithm: it runs through all the different parameters specified into a parameter grid and produces the best combination of parameters, based on a scoring metric. In the Appendix the hyperparameter selection for each classifier can be found.

Ordinal Classifier

In order to try to exploit the order information in the class, an implementation of the approach proposed by (Frank 2001) which is specific for ordinal classification. It is an approach that enables any standard classification algorithm to make use of ordering information in class attributes.

Basically the method consists in transforming a k-class ordinal classification problem into a k-1 binary classification problem: the ordinal attribute with k different values is converted into k-1 binary attributes.

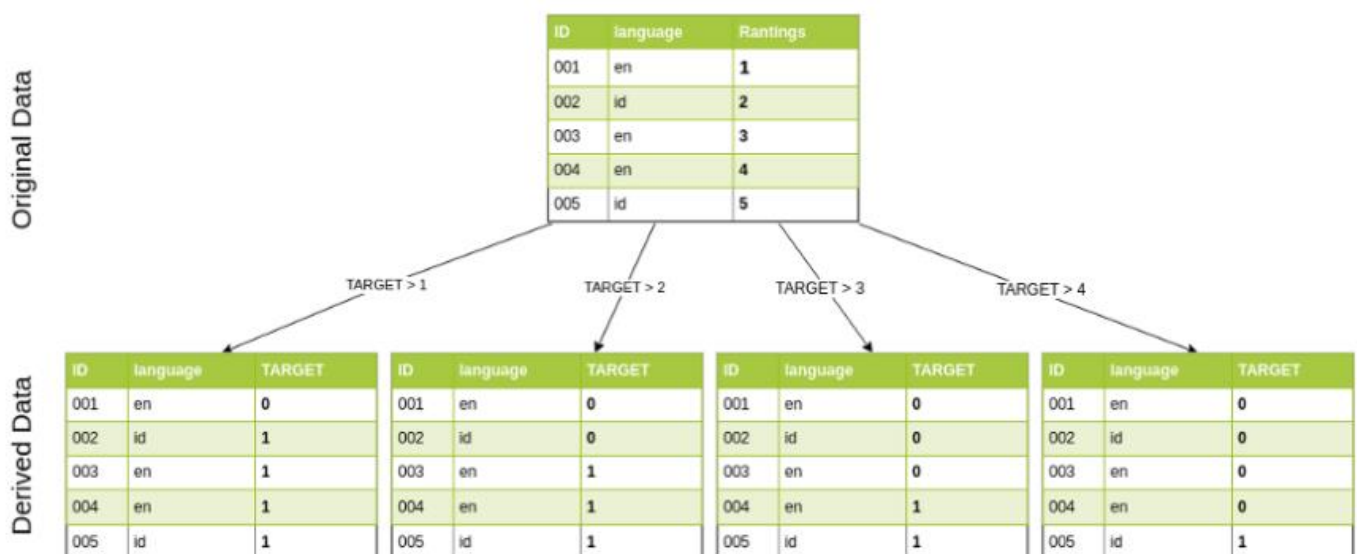


Figure 2 source: <https://towardsdatascience.com/simple-trick-to-train-an-ordinal-regression-with-any-classifier-6911183d2a3c>

k-1 classifiers must be trained and then use them to predict the probabilities of the ordinal value

$$Pr(y=1) = 1 - Pr(\text{Target} > 1)$$

$$Pr(y=2) = Pr(\text{Target} > 1) - Pr(\text{Target} > 2)$$

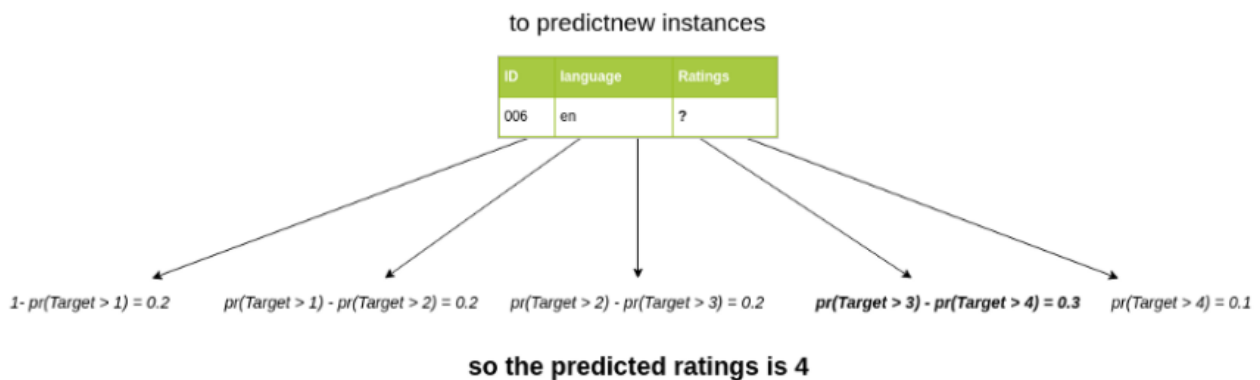
$$Pr(y=3) = Pr(\text{Target} > 2) - Pr(\text{Target} > 3)$$

$$Pr(y=4) = Pr(\text{Target} > 3) - Pr(\text{Target} > 4)$$

$$Pr(y=5) = Pr(\text{Target} > 4)$$

The used classifiers must be able to estimate output class **probability**.

The predicted value is the one associated with the higher probability.



```

class OrdinalClassifier():

    def __init__(self, clf):
        self.clf = clf
        self.clfs = {}

    def fit(self, X, y):
        self.unique_class = np.sort(np.unique(y))
        if self.unique_class.shape[0] > 2:
            for i in range(self.unique_class.shape[0]-1):
                # for each k - 1 ordinal value we fit a binary classification problem
                binary_y = (y > self.unique_class[i]).astype(np.uint8)
                clf = clone(self.clf)
                clf.fit(X, binary_y)
                self.clfs[i] = clf

    def predict_proba(self, X):
        clfs_predict = {k:self.clfs[k].predict_proba(X) for k in self.clfs}
        predicted = []
        for i,y in enumerate(self.unique_class):
            if i == 0:
                # V1 = 1 - Pr(y > V1)
                predicted.append(1 - clfs_predict[y-1][:,1])
            elif y in clfs_predict:
                # Vi = Pr(y > Vi-1) - Pr(y > Vi)
                predicted.append(clfs_predict[y-2][:,1] - clfs_predict[y-1][:,1])
            else:
                # Vk = Pr(y > Vk-1)
                predicted.append(clfs_predict[y-2][:,1])
        return np.vstack(predicted).T

    def predict(self, X):
        return np.argmax(self.predict_proba(X), axis=1) + 1

```

Figure 3 Python implementation of OrdinalClassifier

Feature selection

The dataset after the pre processing phase is composed by 60 attributes, but the aim is also to reduce as much as possible the data that the user has to possess and enter in order to obtain the rating for the player.

This is not only to make the user experience easier, but also because some of this data is available for free online on some sites such as WhoScored.com, but some, such as network analysis data which requires data of all the passes of a team during a match is not easy to find except through private companies who provide accurate statistical data by analysing matches, but this is obviously a higher cost for the user.

So the models have been tested also after having performed a feature selection step.

The feature selection methods tried are:

From scikit-learn:

- **SelectKBest**: removes all but the k highest scoring features. As scoring function was used *fclassif* which estimate the degree of linear dependency between two random variables
- **SelectFromModel**: used alongside any estimator (model) that assigns importance to each feature. Feature are removed if the corresponding importance of the feature values are below the provided threshold

From Weka:

- **CfsSubsetEval + BestFirst**: Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them.
Subsets of features that are highly correlated with the class while having low intercorrelation are preferred.
The using of this method has been decided in order to try a method which evaluates not a single attribute but a collection of attributes, considering dependencies among them.

Performances results

	Attribute selection	Num. features selected	Resampling	Macro avg MAE	Avg Precision	Avg Recall	Avg F1 score	Avg Accuracy	Time
Logistic Regression	SelectFrom Model	30	Random Oversampler	0.305	0.637	0.704	0.657	0.653	0.023
Logistic Regression	None	59	Random Oversampler	0.306	0.643	0.704	0.663	0.66	0.021
SVM	None	59	SMOTE	0.311	0.651	0.702	0.669	0.663	0.201
SVM	SelectFrom Model	30	SMOTE	0.313	0.652	0.699	0.667	0.657	0.216
Logistic Regression	None	59	None	0.314	0.728	0.694	0.706	0.684	0.025
SVM	None	59	None	0.316	0.724	0.693	0.705	0.68	0.251
Logistic Regression	SelectFrom Model	30	None	0.317	0.72	0.691	0.702	0.678	0.021
SVM	SelectFrom Model	30	None	0.324	0.723	0.685	0.7	0.678	0.188
XGBClassifier	SelectFrom Model	30	SMOTE	0.341	0.712	0.672	0.686	0.656	0.154
XGBClassifier	None	59	Random Oversampler	0.342	0.686	0.673	0.676	0.652	0.031
XGBClassifier	CfsSubsetEval + BestFirst	11	None	0.348	0.611	0.671	0.633	0.601	0.02
Logistic Regression	SelectKBest	35	None	0.35	0.718	0.667	0.686	0.658	0.021
XGBClassifier	SelectKBest	35	SMOTE	0.355	0.701	0.663	0.677	0.654	0.031
SVM	CfsSubsetEval + BestFirst	11	None	0.357	0.599	0.662	0.607	0.573	0.147
OrdinalClassifier + DecisionTree	CfsSubsetEval + BestFirst	11	None	0.358	0.532	0.663	0.553	0.541	0.007
Logistic Regression	CfsSubsetEval + BestFirst	11	None	0.363	0.606	0.654	0.621	0.583	0.005
XGBClassifier	SelectKBest	35	None	0.364	0.714	0.652	0.675	0.652	0.031
RandomForest	None	59	Random Oversampler	0.365	0.704	0.649	0.669	0.651	0.048
KNN	CfsSubsetEval + BestFirst	11	None	0.371	0.575	0.654	0.602	0.578	0.292
Random Forest	None	59	None	0.375	0.716	0.641	0.667	0.651	0.047
Random Forest	SelectFrom Model	30	None	0.377	0.707	0.643	0.665	0.644	0.072
Random Forest	SelectKBest	35	None	0.381	0.707	0.638	0.662	0.64	0.068

Random Forest	CfsSubsetEval + BestFirst	11	None	0.387	0.587	0.642	0.607	0.573	0.048
Random Forest	CfsSubsetEval + BestFirst	11	Random OverSampler	0.414	0.596	0.622	0.605	0.573	0.04
XGBClassifier	CfsSubsetEval + BestFirst	11	Random OverSampler	0.436	0.613	0.597	0.6	0.577	0.024
Ordinal classifier + decision tree	None	59	Random OverSampler	0.441	0.633	0.59	0.598	0.587	0.036
Ordinal + Decision Tree	None	59	None	0.469	0.668	0.561	0.587	0.586	0.035
Ordinal classifier + decision tree	CfsSubsetEval + BestFirst	11	Random OverSampler	0.475	0.546	0.582	0.504	0.525	0.011
SVM	CfsSubsetEval + BestFirst	11	SMOTE	0.486	0.598	0.549	0.524	0.527	0.149
KNN	SelectKBest	35	None	0.493	0.678	0.535	0.548	0.594	0.199
KNN	None	59	None	0.519	0.66	0.523	0.529	0.592	0.264
KNN	CfsSubsetEval + BestFirst	11	Random Oversampler	0.575	0.589	0.513	0.53	0.538	0.497

Analysing the results, we can say that the resampling of the training set improved the performances of some models like Logistic Regression, SVM, XGBClassifier and Random Forest, while the feature selection didn't change so much the results.

Logistic Regression model was the one which performed better.

We can also notice that the algorithm *OrdinalRegression* even if is designed for Ordinal classification did not reach good performances, especially if compared with other algorithms.

As we can see from the results, the models do not reach very high values in terms of precision and recall, so the models are not very accurate in predicting the exact class, but as already mentioned the focus was given more to minimising the error in the prediction. An error of 0.3 is acceptable, so we can consider satisfied with the results obtained.

Statistical Significance

We have to understand whether the differences in the results among different models are statistically significant or not, in order to establish which model is best used in the

application. So the paired **Student's t-test** on the macro averaged MAE and F1 score was carried out considering all the results of all the folds of the cross validation.

For the comparison were selected the models which reached the best scores. Also the model XGBClassifier with CfsSubsetEval + BestFirst was tried in the test, since it is the one which reaches the highest scores with the lowest number of features selected

	Macro avg MAE		F1 score	
Logistic regression w/o attribute selection	0.314	$p = 0.718$	0.706	$p = 0.388$
—				
Logistic regression Attr. Selection = <i>SelectfromModel</i>	0.317		0.702	
Logistic regression <i>SelectfromModel</i> + <i>Resampling</i>	0.305	$p = 0.065$	0.657	$p = 0.001$
—				
Logistic regression Attr. Selection = <i>SelectfromModel</i>	0.317		0.702	
XGBClassifier Attr. Selection = <i>CfssubsetEval</i> + <i>BestFirst</i>	0.348	$p = 0.02$	0.633	$p = 1.23e-05$
—				
Logistic regression Attr. Selection = <i>SelectfromModel</i>	0.317		0.702	

$\alpha = 0.05$

The test was performed after verifying the assumption that samples are normally distributed using of the **Shapiro-Wilk** test.

In the Logistic Regression models there is not a difference statistically significant in the MAE.

Instead, the difference with the CfssubsetEval+BestFirst version of XGBClassifier is statistically significant, so this latter cannot be chosen for the application.

Hence the chosen model was the *SelectFromModel* with no resampling version of Logistic Regression, because it shows a better f1 score with a difference statistically significant compared to the other models, considering also the smaller number of attributes required (almost half of the original number of features).

Final considerations

Although for application purposes the model obtained gives acceptable results, it is not very accurate in predicting exact classes.

This may be due to the fact that often in judging a performance, journalists may also be unconsciously influenced by factors beyond the specific match, such as the market value of a player, which when very high tends to bring high expectations to those judging a performance.

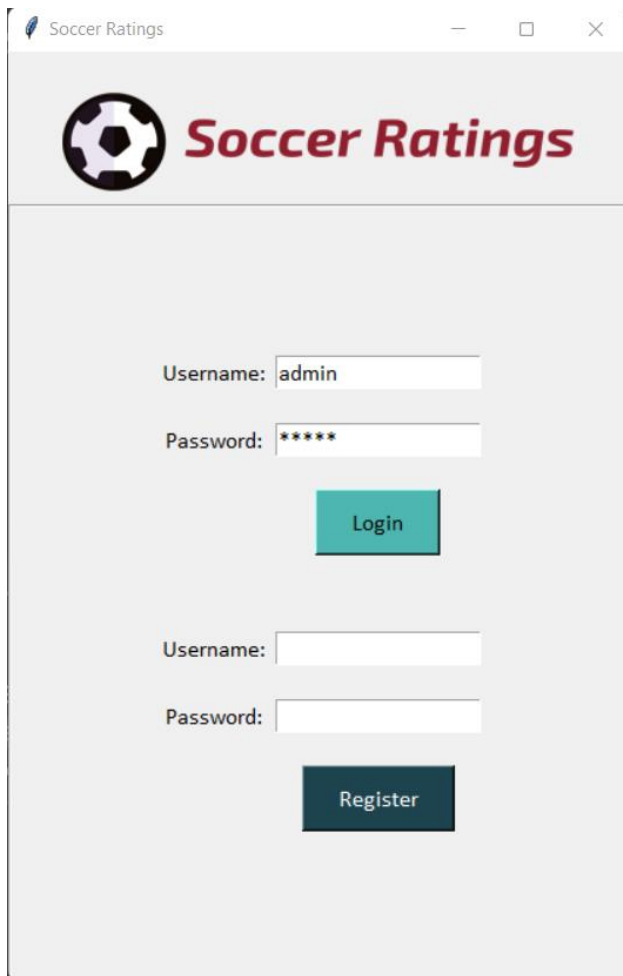
The model could be improved by considering additional attributes (current market value of the player, age, etc.) which can help to better explain the assignment of one rating rather than another.

The dataset was also analysed in (Manfredi 2020), although with slightly different objectives. Comparing the results is difficult, because only the MAE is reported (not macro averaged and as already discussed not suitable for unbalanced datasets) which is between 0.52 and 0.54 according to different position roles.

Moreover, only a part of the dataset is analysed (only human raters). Additionally a different scale is used for the ratings and since the MAE is a scale-dependent accuracy measure it cannot be compared with the results of this work.


Application User Guide

In order to use the application, a registered user must login into the system.



The image shows a web browser window titled "Soccer Ratings". The page features a header with a soccer ball icon and the text "Soccer Ratings" in a red, italicized font. Below the header, there are two sets of login fields. The first set has a "Username:" label followed by a text input containing "admin", and a "Password:" label followed by a text input containing six asterisks. Below these is a teal "Login" button. The second set of fields is identical but empty, with a "Register" button below it.

Soccer Ratings

 **Soccer Ratings**

Username:

Password:

Login

Username:

Password:

Register

Once logged in, the user has access to the main page, where he can select the actions he want to perform:



Soccer Ratings

Insert new players to your team:

Select a role

Insert new player

Select the player to delete

Delete a player


Rate a player

Select the player

View a player's ratings history series

View the starting lineup

In the case the user wants to rate a player, he will have access to another page where he can select the player he wants to rate and he can enter all the data in the respective input field and then click on the button “Compute the rating”:


Soccer Ratings

←

Accurated long passes <input type="text" value="3"/>	Stopped shots <input type="text" value="1"/>	Ground duels won <input type="text" value="10"/>	Aerials won <input type="text" value="3"/>
Possession lost <input type="text" value="7"/>	Clearances <input type="text" value="1"/>	Interceptions <input type="text" value="2"/>	Tackles <input type="text" value="1"/>
Shots on target <input type="text" value="0"/>	Own goals <input type="text" value="0"/>	# actions in which player is involved <input type="text" value="10"/>	# total actions of team <input type="text" value="100"/>
<input type="checkbox"/> Red Cards <input type="checkbox"/> Yellow Cards	<input checked="" type="checkbox"/> Win <input type="checkbox"/> Lost <input checked="" type="checkbox"/> Starter	# actions which end with a shot where player is involved <input type="text" value="0"/>	

Compute the rating

Rating result

4

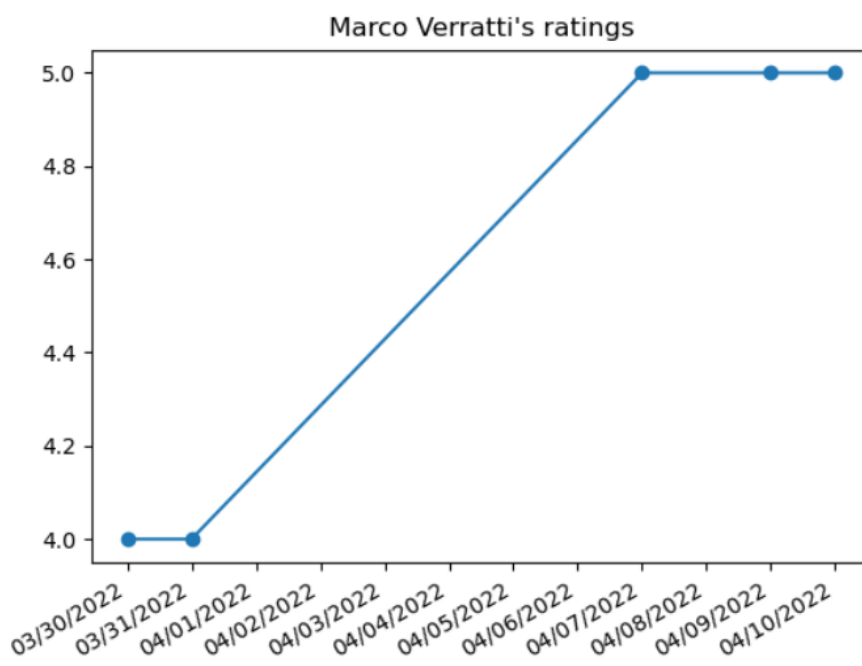
The rating is 4

OK

The user can select among three different types of formation ('4-4-2', '4-3-3' or '3-5-2') and the formation will be displayed to the screen:

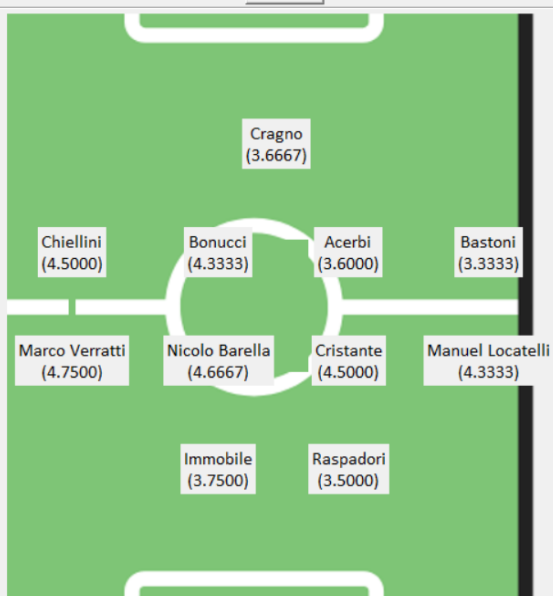
Select the player
⌵

View a player's ratings history series





4-4-2



References

- Frank, E., Hall, M. "A simple approach to ordinal classification." Technical Report 01/05, Department of Computer Science, University of Waikato, 2001.
- Gaudette, L., Japkowicz, N. "Evaluation Methods for Ordinal Classification." By Berlin, Heidelberg Springer. 2009.
- K. Dembczyński, W. Kotłowski, R. Slowiński. *Ordinal classification with decision rules*, in: *Proceedings of the ECML/pkdd '07 Workshop on Mining Complex Data*. Warsaw, PL, 2007.
- Manfredi, G. "Human evaluations of players' performances and the role of networks in soccer." Eindhoven, 2020.
- S. Baccianella, A. Esuli, F. Sebastiani. "Evaluation measures for ordinal regression." *Proceedings of the Ninth International Conference on Intelligent Systems Design and Applications, ISDA'09*. 2009. 283-287.

Appendix

Hyperparameter selection

XGBClassifier:

- max_depth=5
- min_child_weight=3
- colsample_bytree=0.9
- subsample=0.8

KNN:

- n_neighbors= 21
- leaf_size=1
- p=1
- weights="distance"

SVM:

- kernel= 'linear'
- C=0.7
- gamma='auto'

Random Forest:

- max_features=17
- min_samples_split=3
- max_depth=90

Logistic Regression:

- solver='newton-cg'

List of attributes selected in the chosen model

- Goals
- Assists
- Shots on target
- Dribbles successful
- Key passes: passes made by the player that are followed by a shot
- Touches: number of times the player touches the ball during the game
- Passes accurate
- Crosses accurate

- Long Passes Accurate: passes longer than 23 meters: $\text{SQRT}(\Delta X^2 + \Delta Y^2) > 23$
- Ground Duels Won
- Aerials Won: an opponent player is also present in the aerial duel
- Possession Lost: number of times the player lost possession of the ball
- Clearances
- Stopped Shots: number of shots blocked by the player
- Interceptions
- Tackles
- Yellow cards
- Red cards
- Goals Against Outside the Box (GK)
- Goals Against Inside the Box (GK)
- Saves Outside the Box (GK)
- Saves Inside the Box (GK)
- Saved Penalty Kicks (GK)
- Own goals
- Flow centrality: (# actions where the player is involved/# total actions of the team).
An *action* (or play) is a ball possession of the team with at least 2 consecutive successful passes
- Flow success: # actions in which the player is involved that end with a shot/ # total actions of the team, normalized by an average “success ratio” per player position.
(See (Manfredi 2020) to further information on success ratio)
- Win: binary attribute. 1 if the team won.
- Lost: binary attribute. 1 if the team lost.
- Starter: binary attribute. 1 if the player started the match from the first minute.
- Pos_GK. Binary attribute. 1 if the player is a goalkeeper.