INGEGNERIA DELLA CONOSCENZA

AA 19/20

DOCUMENTAZIONE DEL PROGETTO

Michele Messina [676717]
Francesco Zingariello [683004]

m.messina11@studenti.uniba.it
f.zingariello1@studenti.uniba.it

https://github.com/mikkmessi/ICON_Project

Indice

Analisi

L'obiettivo del progetto è quello di realizzare un'applicazione basata su tecniche di AI in grado di predire la performance dei giocatori di Serie A e consigliare i giocatori migliori da schierare la prossima giornata di Fantacalcio.

L'applicazione è sviluppata completamente in linguaggio Python con l'ausilio delle librerie Scikit-learn, Pandas, Seaborn, Matplotlib e dei software Microsoft Excel.

I dataset con le statistiche dei calciatori e delle squadre sono stati recuperati da "Fbref.com", mentre le medie FantaVoto da "Fantacalcio.it"

Per il versionamento è stato utilizzato il repository Github https://github.com/mikkmessi/ICON_Project dove è possibile reperire i dataset, gli script e la documentazione.

Premesse

I dataset sono stati modellati in modo importante per adattarli alle esigenze:

- Per avere risultati ottimali, sarebbe stato più opportuno avere a disposizione i dati per ogni singola giornata di campionato. Dopo varie ricerche infruttuose si è preferito il sito scelto poiché presentava dati precisi e completi anche se cumulativi, non specifici per ogni giornata;
- Il sito forniva più tabelle distinte per ambito calcistico (es. Azioni Difensive, Tiri, Passaggi...). Si è optato per assemblarle in un unico grande dataset dopo un lavoro di studio e scrematura dei parametri. A questo è stato aggiunto il parametro della media fantavoto, per un totale di 1 dataset da usare per il training (26esima giornata) e 2 dataset per predire i risultati della giornata successiva (27esima per predire la 28esima, 28esima per la 29esima);

- Per i portieri la scelta è ricaduta su un dataset distinto, avendo i parametri di valutazione in numero e tipo differenti dal resto dei giocatori.
- Si è creato un file di testo con una rosa di 25 giocatori come simulazione dell'inserimento della squadra dell'utente, da cui estrarre poi la predizione degli 11 giocatori migliori;
- Si pone all'attenzione che l'algoritmo non tiene in considerazione la probabilità che i 25 giocatori scendano effettivamente in campo.

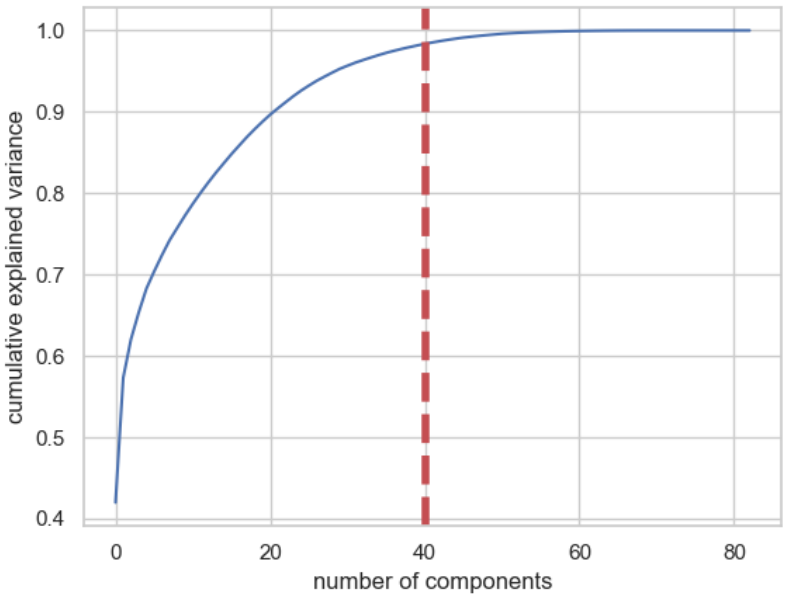
Progettazione

Dopo la fase preliminare di brainstorming e quindi inquadrato l'obiettivo, si è passati quindi alla valutazione e successiva definizione delle scelte progettuali.
L'idea di partenza è stata utilizzare un modello di Regressione Lineare per l'apprendimento supervisionato avente feature-obiettivo la media fantavoto.
Ciascun dataset presentava valori molto diversi, per cui è stato necessario standardizzarli tramite la classe StandardScaler dopo averli "splittati" in train e test set (con train_test_split).
I primissimi test hanno però evidenziato un elevato errore e un test score basso, perciò si è cercato un modello di apprendimento migliore tramite un confronto diretto tra gli score di alcuni modelli selezionati (nello specifico: Linear Regression, Naive Bayes Regressor tramite la classe ARDRegression e Random Forest Regressor) sullo stesso dataset di training.

```
Training e testing giocatori SENZA PCA:

Linear Regression
{'model': LinearRegression(), 'train_score': 0.45798039473383045, 'test_score': 0.4129512573820969, 'media': 0.4354658260579637}
Naive Bayes
{'model': ARDRegression(), 'train_score': 0.4259840396904062, 'test_score': 0.4107060523372156, 'media': 0.4183450460138109}
Random Forest
{'model': RandomForestRegressor(), 'train_score': 0.9251046952868829, 'test_score': 0.8314989914963896, 'media': 0.8783018433916363}
```

Il risultato è stato incoraggiante: nonostante gli score variassero di molto ad ogni test, il Random Forest si è rivelato l'approccio migliore al problema, superando nettamente i concorrenti.
Per tentare di migliorare e stabilizzare l'algoritmo si è pensato di intervenire sui dati riducendone ulteriormente il numero di feature con la tecnica del Principal Component Analysis.
Dopo l'analisi dell'importanza e della varianza nel modello selezionato, visto che con un numero di feature superiore a 40 la varianza cumulata restava pressoché invariata, si è proseguito con l'approssimazione del numero di feature totali a 40.



PCA sul dataset...

	Cumulative Variance Ratio	Explained Variance Ratio
0	0.408736	0.408736
1	0.571912	0.163176
2	0.616656	0.044744
3	0.650950	0.034295
4	0.674906	0.023956
5	0.698451	0.023545
6	0.718665	0.020214
7	0.737546	0.018881
8	0.754502	0.016957
9	0.770032	0.015530
10	0.784111	0.014079
11	0.797817	0.013706
12	0.810562	0.012745
13	0.822603	0.012041
14	0.834014	0.011411
15	0.845286	0.011273
16	0.855975	0.010688
17	0.865880	0.009906
18	0.875109	0.009229
19	0.883997	0.008887

Si è dunque ripetuto il processo di training con questo dataset ridotto, ma i risultati non hanno portato i miglioramenti sperati. Come si può notare, un numero ridotto di feature non influenzava quasi per niente i primi due modelli a discapito del Random Forest, che veniva abbastanza compromesso, pur rimanendo sempre il migliore.

```
Training e testing giocatori CON PCA:

Linear Regression
{'model': LinearRegression(), 'train_score': 0.4318726149818578, 'test_score': 0.40259698111435405, 'media': 0.4172347980481059}
Naive Bayes
{'model': ARDRegression(), 'train_score': 0.4091842930815749, 'test_score': 0.39092441853338533, 'media': 0.4000543558074801}
Random Forest
{'model': RandomForestRegressor(), 'train_score': 0.9319657442350371, 'test_score': 0.5751208363251774, 'media': 0.7535432902801072}
```

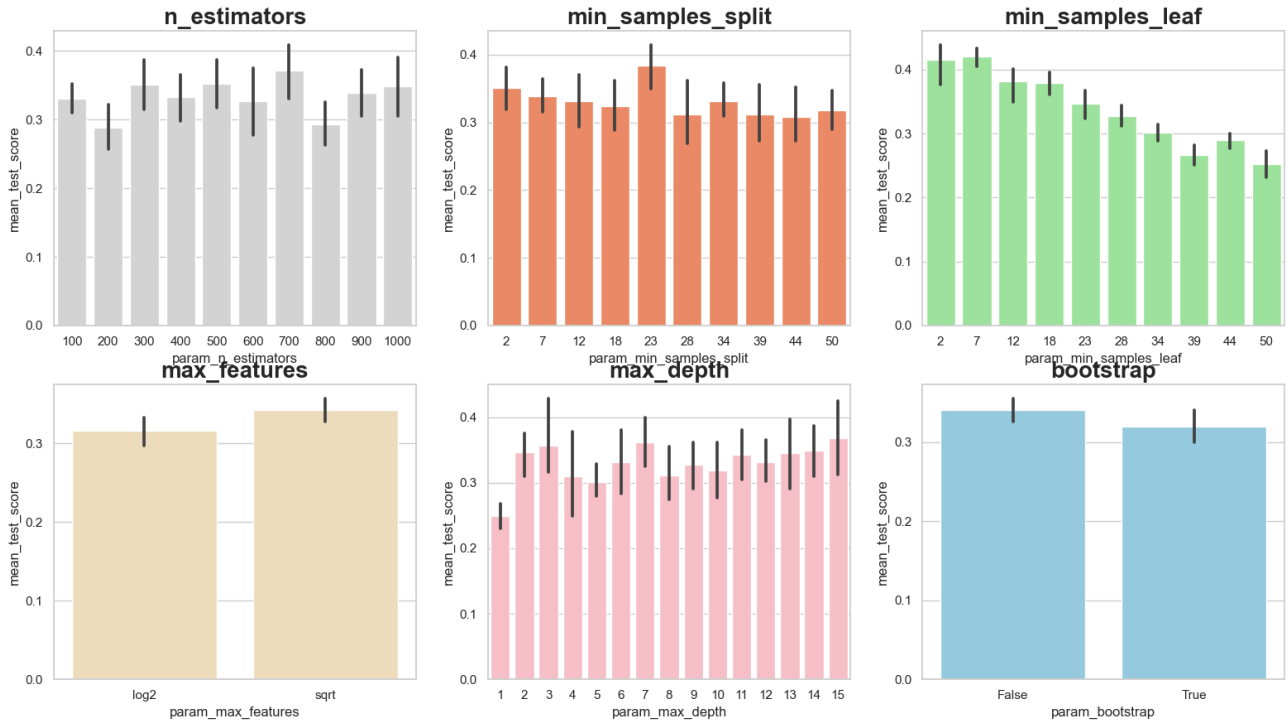
La scelta definitiva è quindi ricaduta sul Random Forest, settando il solo parametro “warm_state” a True, in vista di un ipotetico update del modello con nuovi dati significativi, al momento non disponibili.
Il passo successivo è stato la valutazione dei suoi parametri di funzione (Hypertuning). Ci si è affidati alle funzioni di Scikit-learn RandomizedSearchCV e GridSearchCV per trovare la combinazione di valori dei parametri che restituiva il modello migliore, ripetendo il test anche con il dataset su cui era stata effettuata la PCA. Di seguito i risultati del RandomizedSearchCV, che combina casualmente i valori dei parametri dato un intervallo:

```
Hypertuning RANDOM FOREST SENZA PCA:

Fitting 3 folds for each of 100 candidates, totalling 300 fits
Migliori parametri RandomizedSearchCV:

{'n_estimators': 700, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 11, 'bootstrap': True}
  param_n_estimators param_min_samples_split  ... mean_test_score rank_test_score
0                700                       2  ...        0.450270              1
1                200                      12  ...        0.446686              2
2               1000                      44  ...        0.443706              3
3               1000                      12  ...        0.442778              4
4                700                      18  ...        0.433503              5
5                500                      18  ...        0.432698              6
6                700                      23  ...        0.431363              7
7                400                      23  ...        0.430914              8
8                300                      39  ...        0.428548              9
9                600                      23  ...        0.427865             10
```

Questi sono stati rappresentati anche in forma grafica per permettere di selezionare più facilmente i parametri da usare nel GridSearchCV, che restituisce la combinazione migliore.



Tutti i test effettuati con i relativi score sono riassunti così:

```
SCORE-------------------------------
Model       Train score             Test score
No PCA          0.9251046952868829  0.8314989914963896
PCA             0.9319657442350371  0.5751208363251774
HT RS           0.8279263790128673  0.7790629780086441
HT GS           0.6717050267062294  0.7558021459836524
HT PCA RS       0.8023185908193143  0.5101352554003538
HT PCA GS       0.8071297424806608  0.5099328608251684
END---------------------------------
```

Si evince che tra tutti il modello migliore rimane il Random Forest di default, senza tuning, e con il dataset non ridotto tramite la PCA.

Per quanto riguarda i portieri, si è mantenuto il confronto tra i tre modelli di regressione senza però trovarne uno nettamente superiore a un altro.
L'algoritmo utilizza il migliore risultante ad ogni training.

```
Training and prediction portieri...
Linear Regression
{'model': LinearRegression(), 'train_score': 0.9179674009482335, 'test_score': 0.7871818839740015, 'media': 0.8525746424611175}
Naive Bayes
{'model': ARDRegression(), 'train_score': 0.772232949842656, 'test_score': 0.7764352310311629, 'media': 0.7743340904369094}
Random Forest
{'model': RandomForestRegressor(), 'train_score': 0.9161939399098171, 'test_score': 0.6352142797588707, 'media': 0.7757041098343439}
```

```
Training and prediction portieri...
Linear Regression
{'model': LinearRegression(), 'train_score': 0.9342397390333694, 'test_score': 0.7075637674078058, 'media': 0.8209017532205876}
Naive Bayes
{'model': ARDRegression(), 'train_score': 0.7690730373540003, 'test_score': 0.7930041167364953, 'media': 0.7810385770452478}
Random Forest
{'model': RandomForestRegressor(), 'train_score': 0.942144281319592, 'test_score': 0.336170501676461, 'media': 0.6391573914980265}
```

Una volta appurati i modelli da utilizzare, si è proceduto con l'implementazione dell'algoritmo finale. Leggendo la rosa dei 25 giocatori e la giornata da predire ("28" o "29"), si allena il Random Forest sul dataset ("26") e si prosegue "correggendo" la predizione con alcuni valori dipendenti dalla successiva partita in calendario. Analoga procedura per i portieri, sostituendo il RandomForest con il regressore migliore secondo il confronto. Dopo aver confrontato lo score dei moduli possibili, si ottiene dunque il miglior 11 da schierare.

```
Training giocatori...
Inserire la giornata per cui predire la formazione: 29
Training and prediction portieri...
Linear Regression
{'model': LinearRegression(), 'train_score': 0.9179674009482335, 'test_score': 0.7871818839740015, 'media': 0.8525746424611175}
Naive Bayes
{'model': ARDRegression(), 'train_score': 0.772232949842656, 'test_score': 0.7764352310311629, 'media': 0.7743340904369094}
Random Forest
{'model': RandomForestRegressor(), 'train_score': 0.9161939399098171, 'test_score': 0.6352142797588707, 'media': 0.7757041098343439}

Il modulo migliore per la prossima partita sembrerebbe essere il modulo  3-4-3 con la formazione:

     ID         Nome_Cognome Ruolo  Prediction  Final_weight  Final_score
0     2          Emil-Audero   Por    4.808916        -0.322     4.486916
1   110       Matteo-Darmian   Dif    6.266400         0.548     6.814400
4   255      Gaetano-Letizia   Dif    6.521800         0.070     6.591800
6   455  Leonardo-Spinazzola   Dif    6.324100         0.108     6.432100
15  513      Piotr-Zielinski   Cen    7.144200         0.836     7.980200
9    24        Antonin-Barak   Cen    6.934600         0.285     7.219600
13  371         Ivan-Perisic   Cen    6.250200         0.552     6.802200
11   70     Antonio-Candreva   Cen    7.023300        -0.316     6.707300
20  414    Cristiano-Ronaldo   Att    8.933400         0.559     9.492400
16  360           Joao-Pedro   Att    7.909400        -0.273     7.636400
17  336          MBala-Nzola   Att    7.876800        -0.293     7.583800
```

Gli 11 sono riportati con i relativi valori di:

- *prediction*, esattamente il valore predetto dal regressore;
- *final_weight,* bonus o malus dipendente dalla squadra di appartenenza e avversaria;
- *final_score,* somma aritmetica tra i precedenti.

Implementazione

L'implementazione che ne segue è di facile interpretazione.
Si hanno in totale 5 script: *main.py, line_up.py, main_testing.py, testing.py* e *modeling.py* organizzati come segue:

- In *main_testing.py* si eseguono unicamente le funzioni usate per lo studio dei modelli, contenute in *testing.py* (*best_regressor, principal_component_analysis, hypertuning* e *importance);*

```python
# Creazione dizionario classificatori, che contiene come chiavi il nome dei regressori, come valori un istanza di essi
dict_regressors = {
    "Linear Regression": LinearRegression(),
    "Naive Bayes": ARDRegression(),                    # Automatic Relevance Determination Regression
    "Random Forest": RandomForestRegressor()
}

pca = PCA(n_components=40)


def best_regressor(X_train, Y_train, X_test, Y_test, no_regressors=3):
    '''
        Dati i set di training e test, addestra ogni regressore in dict_regressors, calcola il loro train e test score,
        li stampa per ogni regressore e restituisce il miglior regressore in base alla media tra essi.

    :param      X_train, Y_train, X_test, Y_test: list
    :param      no_regressors: integer

    :return:    best_model: regressor
    '''
    dict_models = {}
    massimo = 0

    for regressor_name, regressor in list(dict_regressors.items())[:no_regressors]:

        regressor.fit(X_train, Y_train)
        train_score = regressor.score(X_train, Y_train)
        test_score = regressor.score(X_test, Y_test)
        mean = (train_score + test_score) / 2

        if mean > massimo:
            massimo = mean
            dict_best_model = {'name': regressor_name, 'model': regressor, 'train_score': train_score, 'test_score': test_score, 'media': mean}

        dict_models[regressor_name] = {'model': regressor, 'train_score': train_score, 'test_score': test_score, 'media': mean}

    for names, values in dict_models.items():
        print(names)
        print(values)

    return dict_best_model
```

- In *modeling.py* si trovano funzioni per la modellazione dei dataset (*load_and_model, split_and_std, get_team);*

```
def split_and_std(stats):
    '''
        Una volta rimossi i parametri letterali dal dataframe, la funzione divide "stats" in train e test set e
        standardizza i valori delle feature.

    :param      stats: pandas dataframe
    :return:    X_train_std, X_test_std, Y_train, Y_test: list
    '''

    stats = stats.drop(['ID', 'Nome_Cognome', 'Ruolo', 'Squadra'], axis=1)

    X = stats[["Partite_giocate", "PG_Titolare",    "Min_giocati", "Min_90", "Reti", "Assist", "Reti_no_rig", "Reti_rig",
               "Rig_tot", "Amm", "Esp", "Reti_90", "Assist_90", "Compl", "Tent", "%Tot", "Dist",    "Dist_prog",
               "Pass_Assist", "Pass_tiro", "Pass_terzo", "Pass_area", "Cross_area", "Pass_prog", "Tocchi", "Drib_vinti",
               "Drib_tot", "%Drib_vinti", "Giocatori_sup", "Tunnel", "Controlli_palla", "Dist_controllo",
               "Dist_controllo_vs_rete", "Prog_controllo_area_avv", "Controllo_area", "Controllo_perso",
               "Contrasto_perso", "Dest", "Ricevuti", "Ricevuti_prog", "Tiri_reti", "Tiri", "Tiri_specchio",
               "%Tiri_specchio", "Tiri_specchio_90", "Goal_tiro", "Dist_avg_tiri", "Tiri_puniz", "Contr", "Contr_vinti",
               "Dribbl_blocked", "Dribbl_no_block", "Dribbl_sub", "%Dribbl_blocked", "Press", "Press_vinti", "%Press_vinti",
               "Blocchi", "Tiri_block", "Tiri_porta_block", "Pass_block", "Intercett", "Tkl_Int", "Salvat", "Err_to_tiro",
               "Azioni_tiro", "Pass_tiro_gioco", "Pass_tiro_no_gioco", "Dribbling_tiro", "Tiri_tiro", "Falli_sub_tiro",
               "Azioni_dif_tiro", "Azioni_gol", "Pass_gol_gioco", "Pass_gol_no_gioco", "Dribbling_gol", "Tiri_gol",
               "Falli_gol", "Azioni_dif_gol", "Azioni_Autogol", "Ruolo_Att", "Ruolo_Dif", "Ruolo_Cen"]].values

    Y = stats["Mf"].values

    # suddividiamo il dataseet in due dataset, uno di training ed uno di test
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)

    # standardizzo il train set creando un modello di standardizzazione
    X_train_std = ss.fit_transform(X_train)
    X_test_std = ss.transform(X_test)

    return X_train_std, X_test_std, Y_train, Y_test
```

- *line_up.py* comprende le funzioni per l'elaborazione dei dati *(final_weight, final_score, best_goalkeeper e best_eleven);*
- L'algoritmo per la predizione è presente nel *main.py* che usufruisce delle funzioni in *modeling.py* e *line_up.py.*

```
rfr = test.RandomForestRegressor(warm_start=True)

dataset = m.load_and_model(str(TRAIN_FB_DAY))

X_train, X_test, Y_train, Y_test = m.split_and_std(dataset)

print("Training giocatori...")
rfr.fit(X_train, Y_train)

next_fb_day = int(input("\nInserire la giornata per cui predire la formazione: "))
current_fb_day = next_fb_day - 1   # ultima giornata di campionato giocata, usata per predire i voti

my_team, my_team_full = m.get_team(str(current_fb_day))                    # 25 players of user's team

prediction_list = list(rfr.predict(my_team))

df_final_score = lu.final_score(my_team_full, prediction_list, next_fb_day, "Dataset_NOPOR.xlsx", sheet_name=str(current_fb_day))

print("Training e prediction portieri...")
df_final_score_gk = lu.best_goalkeeper(TRAIN_FB_DAY, next_fb_day, str(current_fb_day))

best_team, best_module = lu.best_eleven(df_final_score, df_final_score_gk)
```

Conclusioni

Il modello risultante, nonostante i problemi della Premessa, si è rivelato abbastanza accurato, con uno scarto medio tra 0,2 e 0,5 tra le predizioni e la media dei voti effettivi, considerando anche l'alta aleatorietà dello sport.