Assignment 5

Hyperparameter Optimization (HPO)

Introduzione

Il progetto è stato realizzato su google **colab** in **python**, servendosi delle librerie Sklearn, PyGPGO, pandas e mathplotlib.

Analisi del dataset

Il dataset assegnato è “fertility”: https://www.openml.org/d/1473

Esso contiene 100 record con 9 attributi e la classe di appartenenza (binaria)

Purtroppo da una prima analisi è facile notare come il dataset sia molto **sbilanciato** verso il valore positivo della classe da predire.

Questo fattore è importante da **tenere in considerazione** nelle performance che otterremo successivamente, poichè con sbilanciamenti così significativi solitamente i modelli tendono a predirre qualsiasi record con la classe 1 ottenendo una performance dell’88%.

STEP 1

Il primo step consiste nell'effettuare ottimizzazione degli iperparametri per una NN con 2 hidden layer rispettivamente di 4 e 2 neuroni.
I due iperparametri da ottimizzare sono:

- **Learning rate** (numeric in 0.01 - 0.1)
- **Momentum** (numeric in 0.1 - 0.9)

utilizzando due funzioni di acquisizioni differenti, con 20 iterazioni e usando 5 diverse configurazioni iniziali che devono però essere uguali per entrambe le funzioni di acquisizione. Confrontare poi le performance con Grid Search e Random Search.

Definizione dell NN richiesta:

```
def compute_accuracy_SVC(C,gamma):
        clf = MLPClassifier(hidden_layer_sizes=(4, 2), learning_rate_init=C, momentum=gamma, solver='sgd')
        scores = cross_val_score(clf, x, y, cv=10)
        return (scores.mean())
```

Funzioni di acquisizione scelte:

```
# setting the acquisition function
acq1 = Acquisition(mode="ExpectedImprovement")
acq2 = Acquisition(mode="ProbabilityImprovement")
```

Estratto dell'esecuzione dei due modelli definiti come sopra:

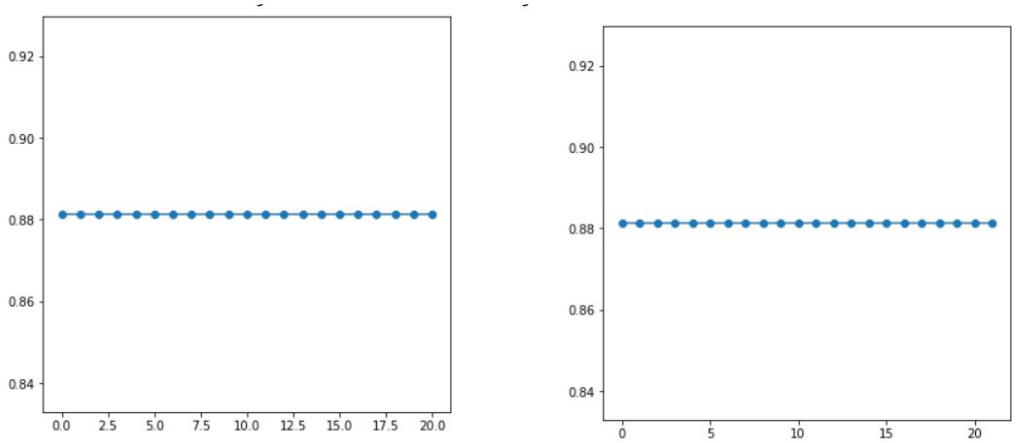
```
Evaluation       Proposed point           Current eval.          Best eval.
init     [0.08732652 0.7931057 ].         0.8814141414141415     0.8814141414141415
init     [0.03350682 0.88651239].         0.8814141414141415     0.8814141414141415
init     [0.0663517  0.13071603].         0.8814141414141415     0.8814141414141415
init     [0.0227548  0.62708556].         0.8814141414141415     0.8814141414141415
init     [0.09327245 0.63575048].         0.8814141414141415     0.8814141414141415
1        [0.01       0.2700279].          0.8814141414141415     0.8814141414141415
2        [0.1        0.27546238].         0.8814141414141415     0.8814141414141415
3        [0.05580856 0.31820736].         0.8814141414141415     0.8814141414141415
4        [0.01 0.1 ].     0.8814141414141415     0.8814141414141415
5        [0.01     0.799983].     0.8814141414141415     0.8814141414141415
6        [0.05082799 0.75722815].         0.8814141414141415     0.8814141414141415
7        [0.1 0.9].       0.8198989898989899     0.8814141414141415
8        [0.0457712  0.71554106].         0.8814141414141415     0.8814141414141415
9        [0.01       0.82132916].         0.8814141414141415     0.8814141414141415
10       [0.06655079 0.63334954].         0.8814141414141415     0.8814141414141415
11       [0.0453656  0.204524531].        0.8714141414141414     0.8814141414141415
```

```
Evaluation       Proposed point           Current eval.          Best eval.
init     [0.04247721 0.22142526].         0.8814141414141415     0.8814141414141415
init     [0.01554675 0.85006988].         0.8814141414141415     0.8814141414141415
init     [0.04244554 0.34600492].         0.8814141414141415     0.8814141414141415
init     [0.02062677 0.36479835].         0.8814141414141415     0.8814141414141415
init     [0.03308245 0.21556132].         0.8814141414141415     0.8814141414141415
1        [0.01579682 0.8084822 ].         0.8814141414141415     0.8814141414141415
2        [0.0323831  0.28327983].         0.8814141414141415     0.8814141414141415
3        [0.03359892 0.28265847].         0.8814141414141415     0.8814141414141415
4        [0.03395296 0.28214409].         0.8814141414141415     0.8814141414141415
5        [0.03409688 0.28177697].         0.8814141414141415     0.8814141414141415
6        [0.01594809 0.78816307].         0.8814141414141415     0.8814141414141415
7        [0.03404039 0.2812452 ].         0.8814141414141415     0.8814141414141415
8        [0.03411003 0.28104798].         0.8814141414141415     0.8814141414141415
9        [0.03414974 0.28090713].         0.8814141414141415     0.8814141414141415
10       [0.03417493 0.28078601].         0.8814141414141415     0.8814141414141415
11       [0.03419041 0.28069175].         0.8814141414141415     0.8814141414141415
```

Risultati finali:

- **Expected Improvement**: 'C', 0.08732652220468878 'gamma', 0.7931057041120937 acc: 0.8814141414141415
- **Probability Improvment**: 'C', 0.04247721352324866), 'gamma', 0.22142526308794946 acc: 0.8814141414141415

Report grafico dell'esecuzione sulle rispettive funzioni di acquisizione:



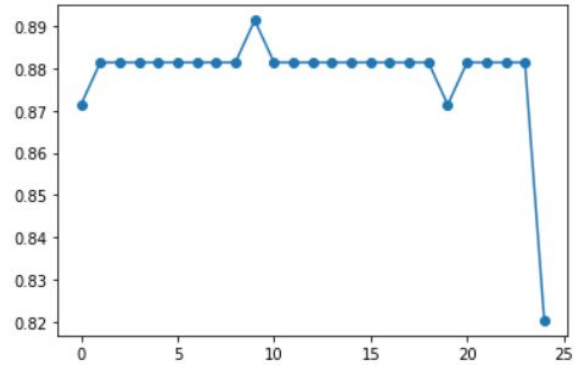
Quindi otteniamo una **accuracy** dell'**88%**.

Viene ora richiesto di confrontare le suddette performance con una ottimizzazione effettuata tramite **Grid Search** e **Random Search**:

GRID SEARCH

```
C_values = np.linspace(xrange[0],xrange[1],5)
gamma_values = np.linspace(yrange[0],yrange[1],5)

res = [0 for n in range(25)]
count = 0
for cc in C_values:
    for gg in gamma_values:
        res[count] = compute_accuracy_SVC( cc, gg )
        count = count+1
plt.plot(res, marker="o")
gridsearch=str(round(max(np.asarray(res)),2))
print("Best on the grid:"+str(round(max(np.asarray(res)),2))+"%%")
```

```
Best on the grid:0.89%%
```



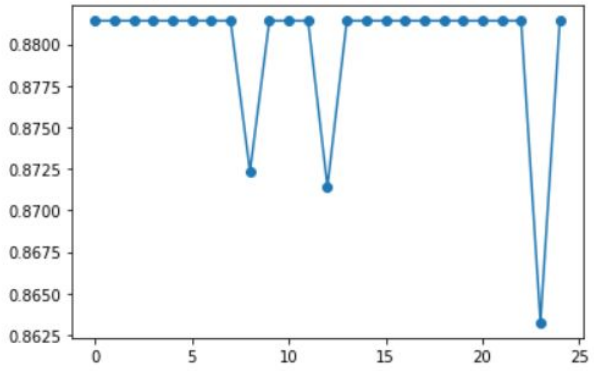
Risultato: **89% di accuracy**

Quindi si rileva che Grid Search ottiene una accuracy leggermente migliore dei due modelli precedenti.

RANDOM SEARCH

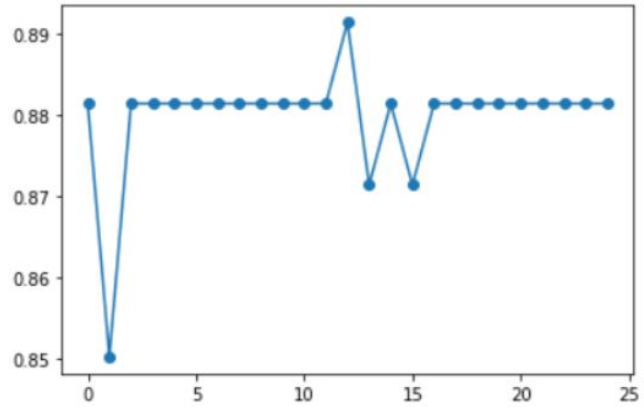
```
import time
import random
res2 = [0 for n in range(25)]
count=0

for i in range (25):
  g1=random.uniform(xrange[0], xrange[1])
  g2=random.uniform(yrange[0], yrange[1])
  res2[count] = compute_accuracy_SVC( g1, g2 )
  count=count+1

plt.plot(res2, marker="o")
randomsearch=str(round(max(np.asarray(res2)),2))

print("Best on the grid:"+str(round(max(np.asarray(res2)),2))+"%%")
```

```
Best on the grid:0.88%%
```

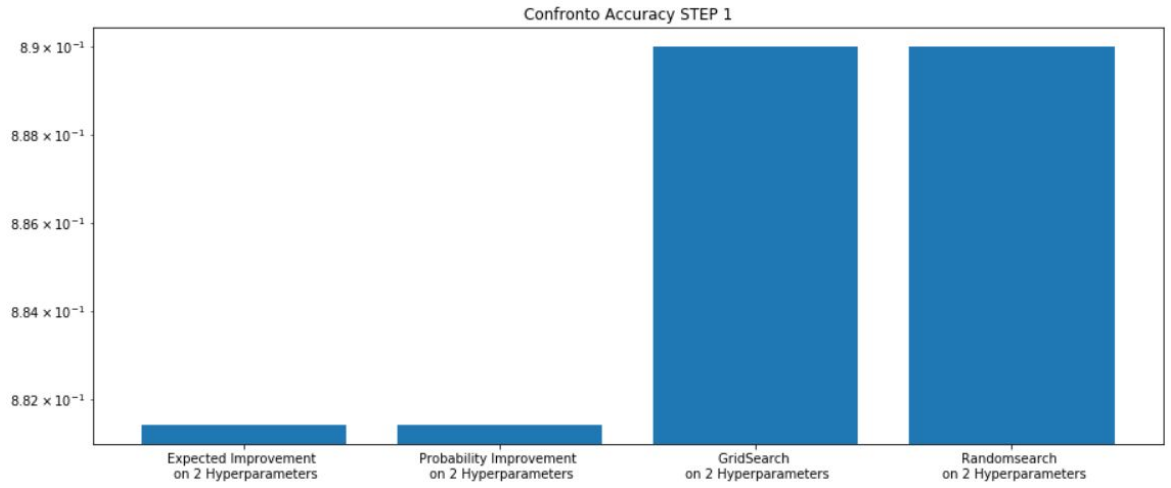


Inizialmente Random Search forniva una accuracy **dell'88%** con sole 25 iterazioni. Rieseguendolo più volte però sono arrivato a picchi di **89%** come per grid search.

```
Best on the grid:0.89%%
```



CONFRONTO PERFORMANCE STEP 1:
Quindi, Grid Search e Random Search arrivano a picchi di **89%** di accuracy contro gli **88%** dei primi modelli analizzati.
C'è da specificare però che Grid Search e Random Search NON sempre arrivano ad **89%**, dipende dai valori che scelgono ad ogni esecuzione mentre i modelli SMBO sono fissi all'**88%** anche dopo diverse esecuzioni



(il grafico è in scala logaritmica)

STEP 2

Il secondo step consiste nell'effettuare ottimizzazione degli iperparametri per una NN con 2 hidden layer con numero di neuroni **da determinare** per ciascuno dei layer (da 1 a 5)
I due iperparametri da ottimizzare sono quindi:

- Learning rate (numeric in 0.01 - 0.1)
- Momentum (numeric in 0.1 - 0.9)
- Numero di Neuroni per Hidden Layer 1
- Numero di Neuroni per Hidden Layer 2

utilizzando due funzioni di acquisizioni differenti, con 100 iterazioni e usando 10 diverse configurazioni iniziali che devono però essere uguali per entrambe le funzioni di acquisizione.

Nuova NN:

```
def compute_accuracy_SVC2(C,gamma,layer1_size, layer2_size):
        clf = MLPClassifier(hidden_layer_sizes=(int(layer1_size), int(layer2_size)), learning_rate_init=C, momentum=gamma, solver='sgd')
        scores = cross_val_score(clf, x, y, cv=10)
        return (scores.mean())
```

Funzioni di Acquisizione scelte:

```
acq1 = Acquisition(mode="ExpectedImprovement")
acq2 = Acquisition(mode="ProbabilityImprovement")
```

Estratto di esecuzione:

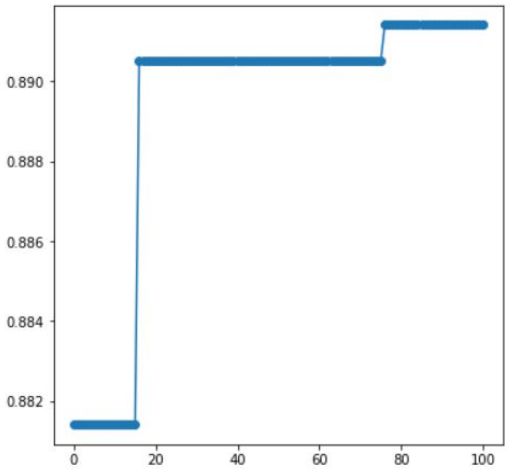
Evaluation	Proposed point	Current eval.	Best eval.	
init	[0.04547813 0.19643327 1.	2.].	0.8814141414141415	0.8814141414141415
init	[0.09119496 0.45791395 2.	2.].	0.8814141414141415	0.8814141414141415
init	[0.02785273 0.7929755 5.	4.].	0.8814141414141415	0.8814141414141415
init	[0.08064327 0.56769181 5.	2.].	0.8814141414141415	0.8814141414141415
init	[0.07271929 0.56048535 1.	2.].	0.8814141414141415	0.8814141414141415
init	[0.03368673 0.47507576 4.	1.].	0.8814141414141415	0.8814141414141415
init	[0.04413337 0.49029427 4.	2.].	0.8814141414141415	0.8814141414141415
init	[0.08230711 0.66229093 1.	4.].	0.8814141414141415	0.8814141414141415
init	[0.02738144 0.11133952 3.	2.].	0.8814141414141415	0.8814141414141415
init	[0.01441173 0.43446815 5.	1.].	0.8814141414141415	0.8814141414141415
1	[0.04828578 0.1178979 1.	5.].	0.8814141414141415	0.8814141414141415
2	[0.06048704 0.14758932 2.	5.].	0.8814141414141415	0.8814141414141415
3	[0.08230803 0.19968319 4.	2.].	0.8814141414141415	0.8814141414141415
4	[0.08899078 0.30200059 5.	4.].	0.8814141414141415	0.8814141414141415
5	[0.06956186 0.68252504 2.	3.].	0.8814141414141415	0.8814141414141415
6	[0.04554256 0.52143728 3.	2.].	0.8814141414141415	0.8814141414141415
7	[0.02754485 0.32150594 5.	2.].	0.8814141414141415	0.8814141414141415
8	[0.01077448 0.49221762 1.	1.].	0.8814141414141415	0.8814141414141415
9	[0.07610045 0.17387177 3.	4.].	0.8814141414141415	0.8814141414141415
10	[0.01587865 0.54279209 5.	5.].	0.8814141414141415	0.8814141414141415
11	[0.05631026 0.20820861 2.	4.].	0.8814141414141415	0.8814141414141415

Evaluation	Proposed point	Current eval.	Best eval.	
init	[0.03285635 0.36137684 5.	2.].	0.8814141414141415	0.8814141414141415
init	[0.05528661 0.28901089 2.	3.].	0.8814141414141415	0.8814141414141415
init	[0.07701634 0.43434352 5.	4.].	0.8814141414141415	0.8814141414141415
init	[0.02824775 0.62434842 1.	3.].	0.8814141414141415	0.8814141414141415
init	[0.04597393 0.83794031 3.	1.].	0.8814141414141415	0.8814141414141415
init	[0.08219028 0.25594631 2.	5.].	0.8814141414141415	0.8814141414141415
init	[0.06575674 0.81942105 3.	5.].	0.8591919191919193	0.8814141414141415
init	[0.08963636 0.82340636 4.	5.].	0.8691919191919192	0.8814141414141415
init	[0.01171793 0.28301562 3.	4.].	0.8703030303030305	0.8814141414141415
init	[0.04146424 0.46994035 2.	1.].	0.8814141414141415	0.8814141414141415
1	[0.05331232 0.4748905 1.	3.].	0.8814141414141415	0.8814141414141415
2	[0.04418955 0.71781408 2.	1.].	0.8814141414141415	0.8814141414141415
3	[0.07483745 0.38584915 2.	3.].	0.8814141414141415	0.8814141414141415
4	[0.03418609 0.44088416 5.	1.].	0.8814141414141415	0.8814141414141415
5	[0.02413084 0.52858188 4.	1.].	0.8814141414141415	0.8814141414141415
6	[0.04757835 0.5423365 1.	2.].	0.8814141414141415	0.8814141414141415
7	[0.06151783 0.65548602 1.	1.].	0.8814141414141415	0.8814141414141415
8	[0.06367345 0.52375695 4.	1.].	0.8814141414141415	0.8814141414141415
9	[0.05430274 0.54982721 5.	3.].	0.8814141414141415	0.8814141414141415
10	[0.05379142 0.55566095 3.	1.].	0.8814141414141415	0.8814141414141415
11	[0.07720556 0.19853092 2.	4.].	0.8814141414141415	0.8814141414141415

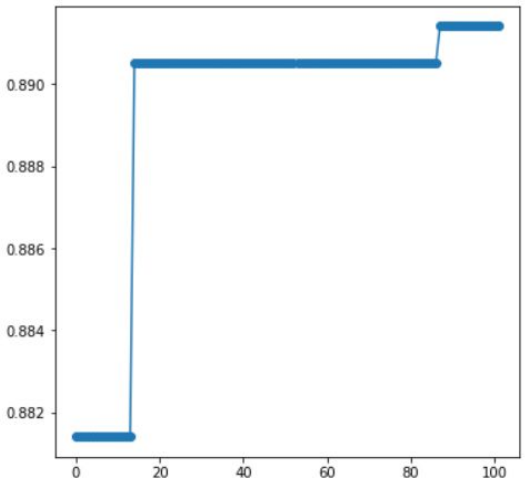
Risultati finali:

- Funzione**: Expected Improvement**:
  - **C**: 0.0925708339868988
  - **gamma**: 0.8298544732361884
  - **layer1_size**: 5.0
  - **layer2_size**: 3.0
  - **acc**: 0.8914141414141415
- Funzione**: Probability Improvement**:
  - **C**: 0.09574774420534946
  - **gamma**: 0.2693325616571006
  - **layer1_size**: 5.0
  - **layer2_size**: 5.0
  - **acc**: 0.8914141414141415

Funzione: **Expected Improvement**

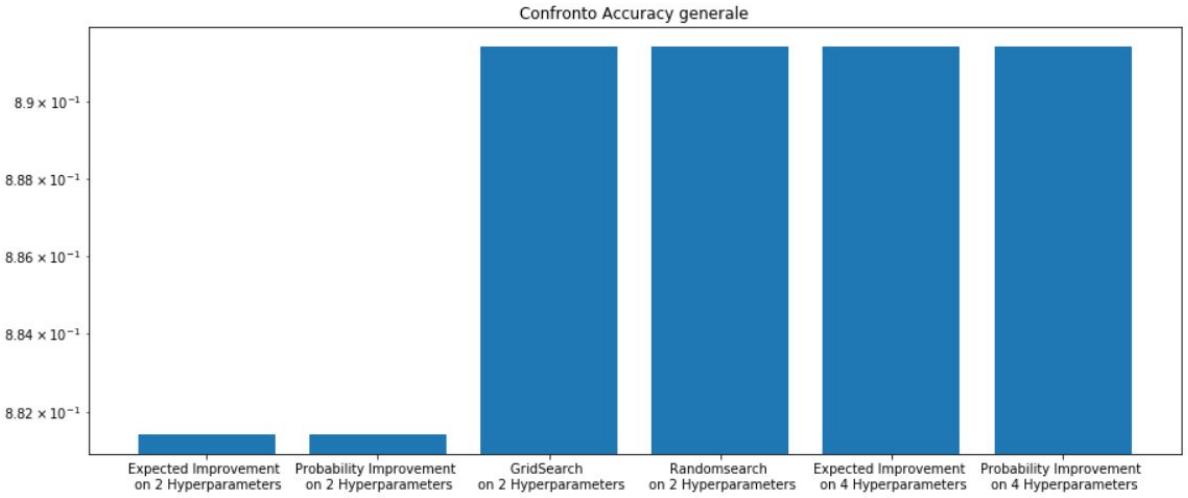


Funzione**: Probability Improvment**:

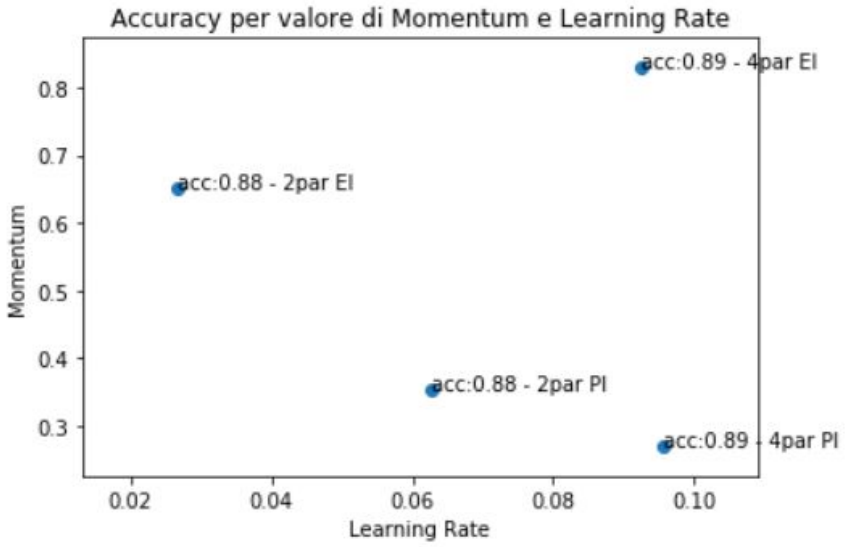


Otteniamo quindi in entrambi i casi un valore di accuracy dell'89%. Superiore a quello individuato precedentemente con l'ottimizzazione su soli 2 parametri.
Dare la possibilità al modello SMBO di ottimizzare anche il numero di Layer sembra dare un vantaggio in termini di performance.

CONFRONTO FINALE ACCURACY con tutti i metodi utilizzati:



CONFRONTO FINALE ACCURACY e valori di learning rate e momentum delle SMBO:



CONCLUSIONI

La soluzione che si è rivelata essere la migliore è la SMBO con 4 parametri che ha sempre dato come performance 89% di accuracy contro l'ottimizzazione con solo 2 parametri che non ha mai superato l'88%.

Grid Seach e Random Search hanno dato risultati incostanti a seconda dell'esecuzione, arrivando comunque ad avere picchi di accuracy all'89% e stando comunque sempre sopra l'88%.

Usare un modello SMBO è sicuramente l’idea migliore in questi casi anche se il dataset assegnato non ha evidenziato in modo marcato questa differenza poichè, come anticipato inizialmente, è molto sbilanciato.