Progetto Data Intensive

A.A.: 2021-2022

Autore: Daniel Guariglia

Matricola: 0000916433

Introduzione

Questo progetto mira a confrontare diversi metodi di Collaborative Filtering paragonando gli errori (MAE e RMSE) ottenuti al fine di identificare il modello più adeguato su questo insieme di dati per predire la valutazione di un utente di un film che non ha valutato.

Il Dataset usato, "The Movies Dataset", raccoglie 26 milioni di valutazioni (da 1 a 5) di 40000 film effettuate da 270000 utenti; per ridurre il tempo di calcolo richiesto nel progetto viene usata una versione ridotta del dataset denominata "_small".

Nel codice vengono mostrate le differenze tra gli approcci User-Based, Item-Based e SVD paragonando i risultati con la predizione casuale della valutazione.

Installazioni

```
#Installazione surprise
!pip install scikit-surprise
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 KB 10.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.21.6)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.7.3)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp38-cp38-linux_x86_64.whl size=2626475 sha256=faacabe4ea647df9
  Stored in directory: /root/.cache/pip/wheels/af/db/86/2c18183a80ba05da35bf0fb7417aac5cddbd93bcb1b92fd3ea
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

```
#installazine Kaggle per scaricare i dataset
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.8/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.8/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.8/dist-packages (from kaggle) (7.0.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.25.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from kaggle) (4.64.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.8/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (4.0.0)
```

Inserimento API Toked di Kaggle (facoltativo)

È possibile scaricare il dataset da Kaggle, per farlo è richiesto l'inserimento di un API Token nella cartella principale del progetto; successivamente eseguire il codice.

In alterantiva leggere il prossimo paragrafo per scaricare da link

```
#!mkdir ~/.kaggle
#!cp kaggle.json ~/.kaggle/
#!chmod 600 ~/.kaggle/kaggle.json
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

Download Dataset

Nome Dataset: The Movies Dataset

movies_metadata.csv: file dei metadati dei film; Contiene informazioni sui film presenti nel set di dati Full MovieLens. Le caratteristiche includono *poster, sfondi, budget, entrate, date di uscita, lingue, paesi di produzione, aziende, titolo del film ecc....*

keywords.csv: contiene le parole chiave della trama del film per i nostri film MovieLens. Disponibile sotto forma di oggetto JSON con stringhe.

credits.csv: contiene informazioni sul cast e sulla troupe per tutti i film. Disponibile sotto forma di oggetto JSON con stringhe.

links.csv: il file che contiene gli ID TMDB e IMDB di tutti i film presenti nel set di dati Full MovieLens.

links_small.csv: contiene gli ID TMDB e IMDB di un piccolo sottoinsieme di 9.000 film del set di dati completo.

ratings_small.csv: il sottoinsieme di 100.000 valutazioni di 700 utenti su 9.000 film presi da ratings.csv.

Tramite API Kaggle

```
#!kaggle datasets download -d rounakbanik/the-movies-dataset
```

```
Downloading the-movies-dataset.zip to /content
100% 227M/228M [00:06<00:00, 41.3MB/s]
100% 228M/228M [00:06<00:00, 36.6MB/s]
```

```
#!unzip the-movies-dataset.zip
```

```
Archive:  the-movies-dataset.zip
  inflating: credits.csv
  inflating: keywords.csv
  inflating: links.csv
  inflating: links_small.csv
  inflating: movies_metadata.csv
  inflating: ratings.csv
  inflating: ratings_small.csv
```

Tramite link

```
import os.path
from urllib.request import urlretrieve
if not os.path.exists("movies_metadata.csv"):
    urlretrieve("https://bit.ly/3Xm3DC4", "movies_metadata.csv")
if not os.path.exists("ratings_small.csv"):
    urlretrieve("http://bit.ly/3iO7Pv3", "ratings_small.csv")
```

Librerie

```
import pandas as pd
import numpy as np
from datetime import datetime
import surprise
from surprise import Dataset
from surprise.model_selection import KFold
from surprise.model_selection import train_test_split
from surprise.model_selection import cross_validate
from surprise.model_selection import GridSearchCV
from surprise.accuracy import rmse
from surprise.accuracy import mae
import matplotlib.pyplot as plt
```

Caricamento dati da CSV

```
#Carico il dataset tramite surprise
csv_reader = surprise.Reader(sep=",", rating_scale=(1, 5), skip_lines=1)
full_dataset = Dataset.load_from_file("ratings_small.csv", csv_reader)
```

```
#Creazione di un trai set e di un test set per validazione Hold-out
trainset, testset = train_test_split(full_dataset, test_size=0.3)
```

Breve panoramica dei dati

In questo paragrafo vene mostrata una analisi dei dati secondo i seguenti punti:

- Numero di voti
- Numero di utenti
- Numero di film
- Valutazione media
- Quanti film ha recensito in media un utente
- I 10 film con la media più alta
- I 10 film più votati
- Generi più presenti

```
movies = pd.read_csv("movies_metadata.csv").\
                    drop(['belongs_to_collection', 'homepage', 'imdb_id', 'poster_path', 'status', 'title', 'video'], axis=1).\
                    drop([19730, 29503, 35587]) # Incorrect data
movies['id'] = movies['id'].astype('int64')
```

```
ratings_df = pd.read_csv('ratings_small.csv')

#Conversione da timestamp a data
ratings_df['date'] = ratings_df['timestamp'].apply(lambda x: datetime.fromtimestamp(x))
ratings_df.drop('timestamp', axis=1, inplace=True)

ratings_df = ratings_df.merge(movies[['id', 'original_title', 'genres']], left_on='movieId',right_on='id', how='left')
ratings_df = ratings_df[~ratings_df['id'].isna()]
ratings_df.drop('id', axis=1, inplace=True)
ratings_df.reset_index(drop=True, inplace=True)

ratings_df.head()
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: DtypeWarning: Columns (1(
  exec(code_obj, self.user_global_ns, self.user_ns)
```

	userId	movieId	rating	date	original_title	genres
0	1	1371	2.5	2009-12-14 02:52:15	Rocky III	[{'id': 18, 'name': 'Drama'}]
1	1	1405	1.0	2009-12-14 02:53:23	Greed	[{'id': 18, 'name': 'Drama'}, {'id': 36, 'name...
2	1	2105	4.0	2009-12-14 02:52:19	American Pie	[{'id': 35, 'name': 'Comedy'}, {'id': 10749, '...
3	1	2193	2.0	2009-12-14 02:53:18	My Tutor	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...

```
#Numero totale di voti
len(ratings_df)
```

```
44994
```

```
#Numero di utenti
ratings_df["userId"].unique().size
```

```
671
```

```
#numero di film
ratings_df["movieId"].unique().size
```

```
2830
```

```
#Valutazione media
ratings_df["rating"].mean()
```

```
3.560985909232342
```

```
#Quanti film ha recensito in media un utente
ratings_df["userId"].value_counts().mean()
```

```
67.05514157973174
```

```
#10 film con la media pià alta
mean_ratings = ratings_df.groupby("original_title")["rating"].mean()
mean_ratings.sort_values(ascending=False).head(10)
```

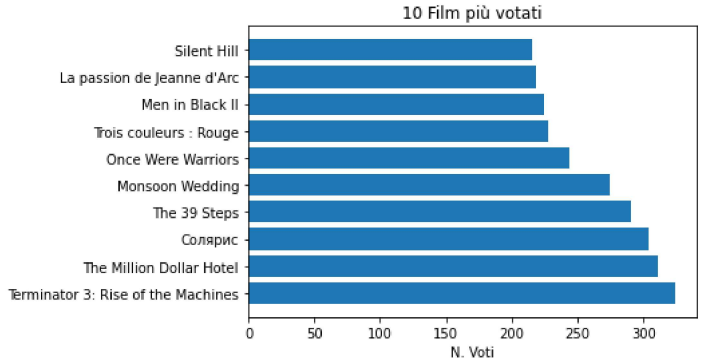
```
original_title
Gentlemen Prefer Blondes                5.0
Anchorman: The Legend of Ron Burgundy   5.0
The Pillow Book                         5.0
```

```
Once Upon a Time in Mexico              5.0
Around the World in Eighty Days         5.0
Nothing Like the Holidays               5.0
De Dominee                              5.0
The Return of Doctor X                  5.0
Ninotchka                               5.0
Night Without Sleep                     5.0
Name: rating, dtype: float64
```

```
#i 10 film con più voti
most_rated = ratings_df.groupby("original_title")["original_title"].count()
most_rated.sort_values(ascending=False, inplace=True)

fig, ax = plt.subplots()
hbars = ax.barh(most_rated.index[:10], most_rated.values[:10])
ax.set_xlabel('N. Voti')
ax.set_title('10 Film più votati')

plt.show()
```



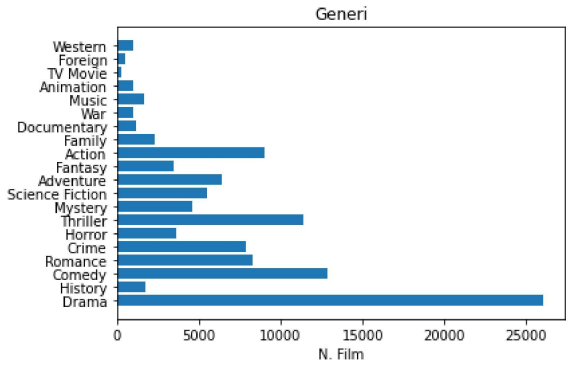

```
#Grafico dei generi più presenti
from ast import literal_eval

ratings_df['genres'] = ratings_df['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name'] for i in x] if isinstance(x, lis

genres_d = {}
for row in ratings_df.values:
  for genres in row[5]:
    if genres in genres_d:
      #esite già la chiave
      genres_d[genres] = genres_d[genres] + 1
    else:
      genres_d[genres] = 1

fig, ax = plt.subplots()
hbars = ax.barh(list(genres_d.keys()), list(genres_d.values()), align='center')
ax.set_xlabel('N. Film')
ax.set_title('Generi')


plt.show()
```



Surprise

Previsioni User Based con KNNBasic

k = numero di utenti simili da considerare

Imposto tramite sim_options l'uso della similarità coseno

In questo paragrafo vengono creati e testati modelli di recommendation user-based variando manualmente k e misura di similarità degli utenti. Viene usato Hold-out come metodo di validazione, i modelli vengono addestrati su un sottoinsieme del dataset chiamato "trainset" e validati sulla restante parte chiamata "testset"; i risultati vengono mostrati per confronto.

Modello con misura di similarità Coseno

```
knn_model_cos = surprise.KNNBasic(k=trainset.n_users, sim_option={"name", "cosine"})
knn_model_cos.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBasic at 0x7f274f9e1b80>
```

Modello con misura di similarità Pearson

```
knn_model_pearson = surprise.KNNBasic(k=trainset.n_users, sim_option={"name", "pearson"})
knn_model_pearson.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBasic at 0x7f2756a4c1f0>
```

Modelli variando K (utenti simili)

```
knn_model_cos_k = surprise.KNNBasic(k=10, sim_option={"name", "cosine"})
knn_model_cos_k.fit(trainset)

knn_model_pearson_k = surprise.KNNBasic(k=10, sim_option={"name", "pearson"})
knn_model_pearson_k.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBasic at 0x7f276772f340>
```

Validazione e Confronto Hold-Out

Sono già stati estratti in precedenza i sottoinsiemi di dati da usare nella validazione tramite Hold-out

```
#Modello con similarità coseno
predictions_cos = knn_model_cos.test(testset)
surprise.accuracy.rmse(predictions_cos), surprise.accuracy.mae(predictions_cos)
```

```
RMSE: 0.9832
MAE:  0.7570
(0.9832091152440641, 0.7570369154937431)
```

```
#Modello con similarità pearson
predictions_pearson = knn_model_pearson.test(testset)
surprise.accuracy.rmse(predictions_pearson), surprise.accuracy.mae(predictions_pearson)
```

```
RMSE: 0.9832
MAE:  0.7570
(0.9832091152440641, 0.7570369154937431)
```

```
#Modello con similarità coseno e K = 10
predictions_cos_k = knn_model_cos_k.test(testset)
surprise.accuracy.rmse(predictions_cos_k), surprise.accuracy.mae(predictions_cos_k)
```

```
RMSE: 0.9726
MAE:  0.7424
(0.9725517352598703, 0.7423516095429743)
```

```
#Modello con similarità pearson e K = 10
predictions_pearson_k = knn_model_pearson_k.test(testset)
surprise.accuracy.rmse(predictions_pearson_k), surprise.accuracy.mae(predictions_pearson_k)
```

```
RMSE: 0.9726
MAE:  0.7424
```

```
(0.9725517352598703, 0.7423516095429743)
```

- in questo caso le metriche di misurazione dell'errore non variano al variare del metodo usato per calcolare la similarità a parità di K
- è possibile osservare un leggero miglioramento riducendo il numero di utenti simili da considerare (k)

Ricerca iperparametri migliori per KNNBasic

Finora sono stati impostati a mano nei modelli gli iperparametri da usare, in questo paragrafo vengono ricercati gli iperparametri che generano l'errore minore da usare nei modelli User-Based. Viene usata K-Fold come validazione.

```
#Divisione set tramite K-Fold
k_fold = KFold(n_splits=3, shuffle=True)
```

```
#Valori per gli iperparametri da testare
grid = {
    "k": [10, 20, 50, trainset.n_users],
    "sim_options": {"name": ["cosine", "pearson"]}
}
```

```
#Addestramento dei modelli User-Based per ricercare gli iperparametri migliori
gs_knnb = GridSearchCV(surprise.KNNBasic, grid, cv=k_fold, refit=True)
gs_knnb.fit(full_dataset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
#Parametri migliori trovati
gs_knnb.best_params
```

```
{'rmse': {'k': 50, 'sim_options': {'name': 'cosine', 'user_based': True}},
 'mae': {'k': 20, 'sim_options': {'name': 'cosine', 'user_based': True}}}
```

```
#Valori RMSE e MAE usando gli iperparametri migliori
best_preds_ub = gs_knnb.test(testset)
surprise.accuracy.rmse(best_preds_ub), surprise.accuracy.mae(best_preds_ub)
```

```
RMSE: 0.8599
MAE:  0.6559
(0.8598969510512633, 0.6558885681073375)
```

- Possiamo osservare un miglioramento rispetto agli errori visti in precedenza

Previsioni User Based con KNNWithMeans

Stima del voto tramite la media degli scostamenti dalle rispettive medie di altri utenti

```
knnWM_model_p = surprise.KNNWithMeans(k=50, sim_option={"name", "cosine"})
knnWM_model_p.fit(trainset)
predictions_knnWM_p = knnWM_model_p.test(testset)
surprise.accuracy.rmse(predictions_knnWM_p), surprise.accuracy.mae(predictions_knnWM_p)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9305
MAE:  0.7131
(0.9305113219155058, 0.7131252306746383)
```

- Tramite l'uso di KNNWithMeans a parità di iperparametri possiamo osservere un leggero peggioramento rispetto al risultato ottenuto con KNNBasic sopra

Ricerca Iperparametri migliori per KNNWithMean

```
#Addestramento dei modello con KNNWithMean per ricerca di iperparametri
gs_knnwm = GridSearchCV(surprise.KNNWithMeans, grid, cv=k_fold, refit=True)
gs_knnwm.fit(full_dataset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
#Parametri migliori trovati
gs_knnwm.best_params
```

```
{'rmse': {'k': 50, 'sim_options': {'name': 'cosine', 'user_based': True}},
 'mae': {'k': 50, 'sim_options': {'name': 'cosine', 'user_based': True}}}
```

```
#Valori RMSE e MAE usando gli iperparametri migliori
best_preds_ub = gs_knnwm.test(testset)
surprise.accuracy.rmse(best_preds_ub), surprise.accuracy.mae(best_preds_ub)
```

```
RMSE: 0.7901
MAE:  0.5964
(0.7901254044125287, 0.5963836277240149)
```

- Otteniamo un ulteriore miglioramento degli errori

Previsioni Item-based

Con la collaborative filtering item-based il voto di un prodotto p viene previsto come media pesata delle valutazioni di quell'utente ai prodotti più simili a p.

La similarità tra p e gli altri oggetti viene calcolata dai rispettivi voti dati dagli utenti che hanno recensito entrambi.

Nel codice presentato viene usato k = 10

```
#Item based con similarità coseno
knn_item_based_cos = surprise.KNNBasic(k=10, sim_options={"name": "cosine", "user_based": False})
knn_item_based_cos.fit(trainset)
#similarità di Pearson
knn_item_based_p = surprise.KNNBasic(k=10, sim_options={"name": "pearson", "user_based": False})
knn_item_based_p.fit(trainset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBasic at 0x7f27552cbd30>
```

Validazione e confronto Hold-Out

```
#RMSE e MAE di Item based con Coseno
pred_knnItemB_c = knn_item_based_cos.test(testset)
surprise.accuracy.rmse(pred_knnItemB_c), surprise.accuracy.mae(pred_knnItemB_c)
```

```
RMSE: 1.0447
MAE:  0.8140
(1.0447385144467112, 0.8139579000362853)
```

```
#RMSE e MAE di Item based con Pearson
pred_knnItemB_p = knn_item_based_p.test(testset)
surprise.accuracy.rmse(pred_knnItemB_p), surprise.accuracy.mae(pred_knnItemB_p)
```

```
RMSE: 1.0229
MAE:  0.7953
(1.0229302303045607, 0.7953250254174548)
```

- Possiamo osservare come in questo caso i modelli item-based producano un errore maggiore rispetto ai modelli user-based.

Ricerca Iperparametri migliori per modelli Item-Based

```
#Ridefinisco grid inserenso user_based=False
grid = {
    "k": [50, trainset.n_users],
    "sim_options": {"name": ["cosine", "pearson"], "user_based": [False]},
}
```

```
#Addestramento dei modello con KNNWithMean per ricerca di iperparametri
gs_knnIB = GridSearchCV(surprise.KNNBasic, grid, cv=k_fold, refit=True)
```

```
gs_knnIB.fit(full_dataset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

```
#Parametri migliori trovati
gs_knnIB.best_params
```

```
{'rmse': {'k': 671, 'sim_options': {'name': 'cosine', 'user_based': False}},
 'mae': {'k': 671, 'sim_options': {'name': 'cosine', 'user_based': False}}}
```

```
#Valori RMSE e MAE usando gli iperparametri migliori
best_preds_ib = gs_knnIB.test(testset)
surprise.accuracy.rmse(best_preds_ib), surprise.accuracy.mae(best_preds_ib)
```

```
RMSE: 0.9385
MAE:  0.7312
(0.9385145034121481, 0.731209358367444)
```

- Otteniamo un peggioramento rispetto agli errori delle recommendation User-Based visi sopra, possiamo concludere che in questo caso sono da preferire approcci User-Based rispetto a quelli Item-Based

Collaborative filtering SVD

In questo paragrafo viene mostrato un approccio tramite Single Value Decomposition, prima con n_factors=10 e successivamente ricercando l'ipermparametro migliore da usare tramite GridSearch

```
svd = surprise.SVD(n_factors=10)
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f27552cb6a0>
```

```
#Test SVD con Hold-out
predictions_svd = svd.test(testset)
surprise.accuracy.rmse(predictions_svd), surprise.accuracy.mae(predictions_svd)
```

```
RMSE: 0.8923
MAE:  0.6891
(0.8922825020989508, 0.6891380271370855)
```

Ricerca Iperparametri migliori per SVD

```
#Definisco una nuova grid a per variare l'unico iperparametro impstato
grid = {"n_factors":  range(10, 100, 10)}
gs_svd = GridSearchCV(surprise.SVD, grid, cv=k_fold, refit=True)
gs_svd.fit(full_dataset)
```

```
gs_svd.best_params
```

```
{'rmse': {'n_factors': 10}, 'mae': {'n_factors': 10}}
```

```
best_preds_svd = gs_svd.test(testset)
surprise.accuracy.rmse(best_preds_svd), surprise.accuracy.mae(best_preds_svd)
```

```
surprise.accuracy.rmse(best_preds_svd), surprise.accuracy.mae(best_preds_svd)
```

```
RMSE: 0.8165
MAE:  0.6285
(0.8164981816659401, 0.628504331478189)
```

Predizione Casuale

Verrà usata per confrontate gli errori ottenuti nel capitolo successivo

```
from surprise import NormalPredictor
rnd = NormalPredictor()
rnd.fit(trainset)
preds_rnd = rnd.test(testset)
rmse(preds_rnd), mae(preds_rnd)
```

```
RMSE: 1.4398
MAE:  1.1485
(1.4398303792415583, 1.1484509850239335)
```

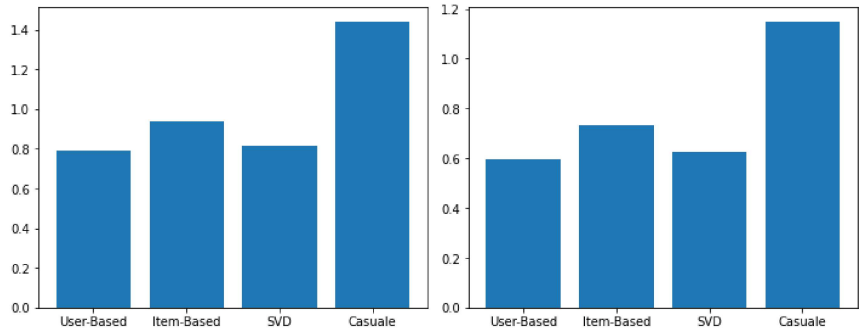
Confronto valori ottenuti

```
import matplotlib.pyplot as plt
approcci = ["User-Based", "Item-Based", "SVD", "Casuale"]
#User_Based KNNWithMean - Item-Based - SVD - Casuale
RMSE = [rmse(best_preds_ub), rmse(best_preds_ib), rmse(best_preds_svd), rmse(preds_rnd)]
MAE = [mae(best_preds_ub), mae(best_preds_ib), mae(best_preds_svd), mae(preds_rnd)]
x_pos = np.arange(len(approcci))

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
axes[0].bar(approcci, RMSE)
axes[1].bar(approcci, MAE)

fig.tight_layout()
```

```
RMSE: 0.7901
RMSE: 0.9385
RMSE: 0.8145
RMSE: 1.4398
MAE:  0.5964
MAE:  0.7312
MAE:  0.6272
MAE:  1.1485
```



Conclusioni

Analizzando gli errori prodotti dei diversi approcci utilizzati si può osservare come le predizioni tramite approccio Item-Based usando KNNWithMean siano le più accurate su sul dataset "_small" utilizzato. Si osservano anche in secondo luogo errori sufficientemente bassi tramite SVD, mentre gli approcci Idem-Based producono errori maggiori ma comunque non paragonabili alla predizione casuale.

Concludo asserendo che dai risultati ottenuti sul dataset usato "_small" tra gli i metodi testati più adeguato alla sia l'Item-Based.

✓ 0 s data/ora di completamento: 16:04