Università degli studi “Roma Tre”

Facoltà di Ingegneria
Corso di laurea in Ingegneria Informatica
A.A. 2019-2020

Progetto Machine Learning

“SISTEMA DI APPRENDIMENTO AUTOMATICO BASATO SULLO STOCHASTIC GRADIENT DESCENT, CHE SUGGERISCE ARTISTI MUSICALI”

Studente:

Daniele Santino matricola 496958

Referente:

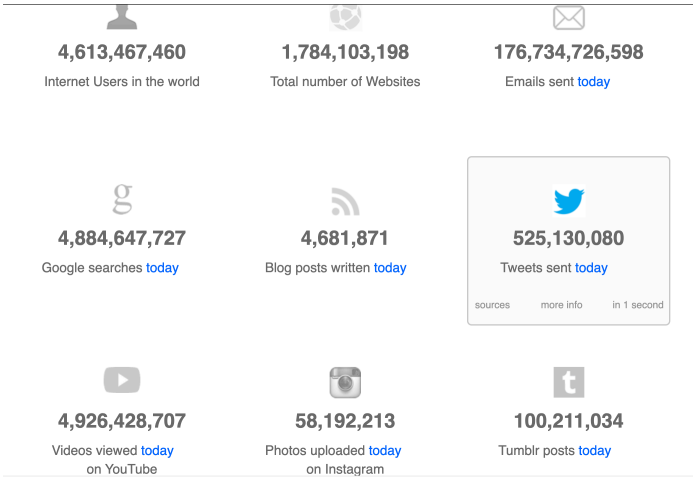
Prof. Giuseppe Sansonetti

Lo scopo di questo progetto è di creare un *Reccomender System* (nel seguito RS) in Python sfruttando le nozioni apprese grazie ai corsi di Sistemi Intelligenti per Internet e Machine Learning .

Il sistema genera una matrice di ratings[1] dal dataset di **last.fm** e pre elabora i dati.

Il modello viene allenato utilizzando lo **Stochastic Gradient Descent** e consente di consigliare dei musicisti da ascoltare.

I RS in una società tecnologicamente evoluta come la nostra, sono diventati fondamentali, perché il numero di informazioni che viaggiano su internet è enorme ed i dati che possono essere ricavati subiscono una crescita esponenziale.



INTERAZIONI SU INTERNET FONTE HTTPS://WWW.INTERNETLIVESTATS.COM/

Gli utenti richiedono sempre più contenuti simili ai propri interessi e ai propri gusti. Si cerca il libro preferito, il prodotto utile per la casa, il film più simile ai propri interessi e si cercano anche le persone più simili al proprio carattere sui social network.

Inoltre si pretende anche una interazione più rapida con i prodotti che utilizziamo quotidianamente.
Ad esempio vorremo acquistare con un semplice click, il detersivo per i panni, non appena questo è finito (ne sono un esempio gli Amazon Dash Button[2])

Vista la la crescita esponenziale dei dati e delle informazioni, alcuni sistemi di calcolo potrebbero risultare inadeguati a compiere le elaborazioni in tempi efficienti.

---

[1] Matrice che esprime l'interazione di un utente verso un determinato item, chiamata matrice di valutazione quando gli utenti esprimono una valutazione esplicita di un elemento.

[2] Amazon Dash è un servizio di ordinazione di beni di consumo che utilizza dispositivi e API proprietari per ordinare beni su Internet.

Per questo motivo si cercano tecniche alternative, basate sul Machine Learning, che in parte ho cercato di riutilizzare in questo progetto.

INDICE

1 - Strumenti utilizzati

Come linguaggio di programmazione si è scelto **Python**, un linguaggio di più “alto livello” rispetto alla maggior parte degli altri linguaggi.

È orientato agli oggetti, ma non in maniera ferrea come ad esempio Java; adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e System testing.

Come software tools si è scelto **Google Colaboratory** (nel seguito colab), un servizio di calcolo online, basato su schede Tesla K80 GPU, le quali presentano una 2496 cuda core che è 45 volte più performante rispetto ad una piattaforma non-GPU.

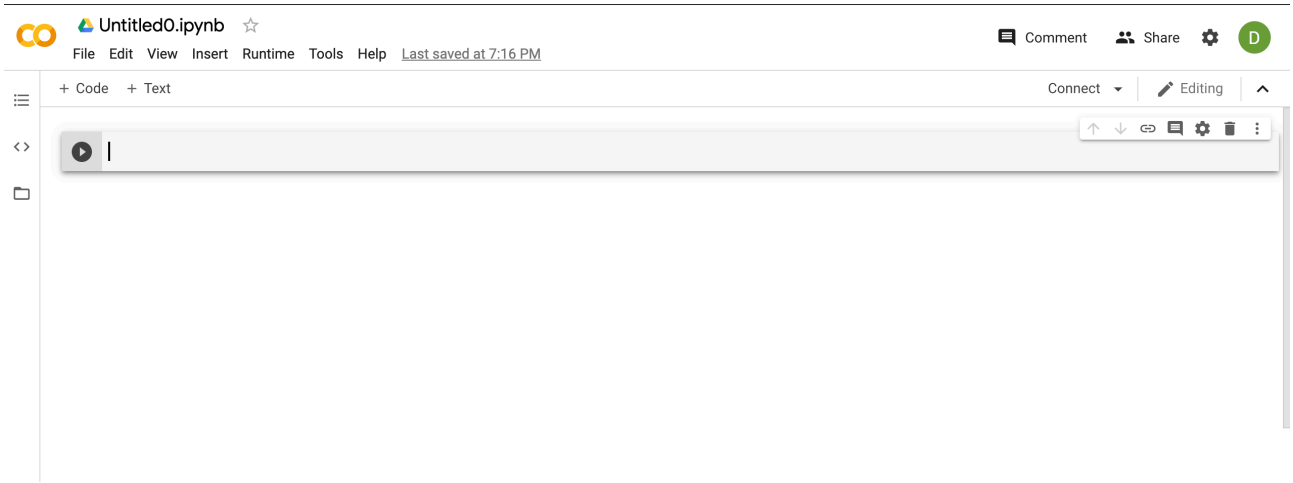
Colab supporta il multi-ambiente, tra i quali:
- TensorFlow
- Keras
- PyTorch
- OpenCV

Interfaccia come Jupiter e di default a runtime abbiamo una versione di Python 2 ma dal menù del runtime è possibile andare a cambiare queste impostazioni.

Inoltre, permette di trasferire l’esecuzione in locale, sul proprio calcolatore qualora le elaborazioni dovessero risultare molto lunghe.

I file vengono salvati su Google Drive e dallo stesso Google Drive è possibile andare a creare un nuovo “colaboratory” : New -> More -> Colaboratory.

Grazie a questo servizio in cloud è possibile andare a sfruttare tutta la potenza di calcolo fornita da Google.



INTERFACCIA DI COLAB

2 - Valutazione degli utenti e dataset

In genere i RS sono basati sulle valutazioni che gli utenti forniscono a proposito di un determinato item.
Un classico esempio è quello delle recensioni dei prodotti su Amazon.

Nel progetto è stata utilizzata una metrica alternativa che nel seguito verrà illustrata per effettuare le raccomandazioni.
In particolare è stato utilizzato un dataset, ovvero una collezione di dati.

Il dataset proviene da last.fm (scaricabile da qui), ospitato da **GroupLens[3].**

Scaricandolo, si hanno a disposizione, tra le altre, 943.347 brani, 505.216 tracce con almeno un tag, 584.897 tracce con almeno una traccia simile.
Inoltre si hanno a disposizione due tipi di dati a livello di brano: tag e brani simili

Contiene vari file, tra cui:

- **user_artists.dat:** userID, artistID, weight RAPPRESENTAZIONE ARTISTA PER UTENTE.
- **artists.dat:** id, name, url, pictureURL.
- **tags.dat:** tagID, tagValue.
- **user_tagged_artists.dat:** userID, artistsID, tagID, day, month, year.
- **user_tagged_artists_timestamp.dat:** userID, artistID, tagID, timestamp.
- **user_friend.dat:** userID, friendID RELAZIONI UTENTE/AMICO.

Il focus è stato posto su user_artists.dat e artists.dat in quanto contengono tutti i dati necessari per fornire consigli per i nuovi artisti musicali a un utente.

Invece per le valutazioni è stato usato il conteggio delle riproduzioni di un utente per ciascun artista.

[3] **GroupLens Research** is a human–computer interaction research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems and online communities.

3 - Caricamento dei dati

I dati sono stati caricati nel Dataframe di ***Pandas:***

Pandas è una libreria che è stata trattata nei corsi universitari che hanno stimolato il progetto.
Serve per manipolare i dati in formato sequenziale o tabellare, quali serie temporali o dati di microarray.

Caratteristiche principali di Pandas sono:

- Caricamento e salvataggio di formati standard per dati tabellari, quali CSV, TSV, file Excel e formati per database.
- Semplicità nell'aggregazione e nell'indicizzazione dei dati.
- Semplicità per l'esecuzione di operazioni numeriche e statistiche.
- Semplicità nella visualizzazione dei risultati delle operazioni.

Pandas in Colab si importa come qualsiasi altro modulo e la sintassi comune è:

Import pandas *as pd*

```
[ ]  riproduzioni = pd.read_csv('/content/drive/My Drive/Colab Notebooks/lastfm/user_artists.dat', sep='\t')
     artisti = pd.read_csv('/content/drive/My Drive/Colab Notebooks/lastfm/artists.dat', sep='\t', usecols=['id','name'])
```

4 - Pulizia dei dati

Prima di lavorare con i dati, andiamo a manipolarli, per rendere più semplice il lavoro sul codice.

```
[ ]  ap = pd.merge(
       artisti, riproduzioni,
       how="inner",
       left_on="id",
       right_on="artistID"
     )
```

```
[ ]  ap = ap.rename(columns={"weight": "numeroRiproduzioni"})
```

Uniamo gli artisti e le riproduzioni degli utenti e rinominiamo la colonna in “numeroRiproduzioni”.

Gli artisti vengono dunque classificati in base al numero di riproduzioni che hanno ricevuto dagli utenti:

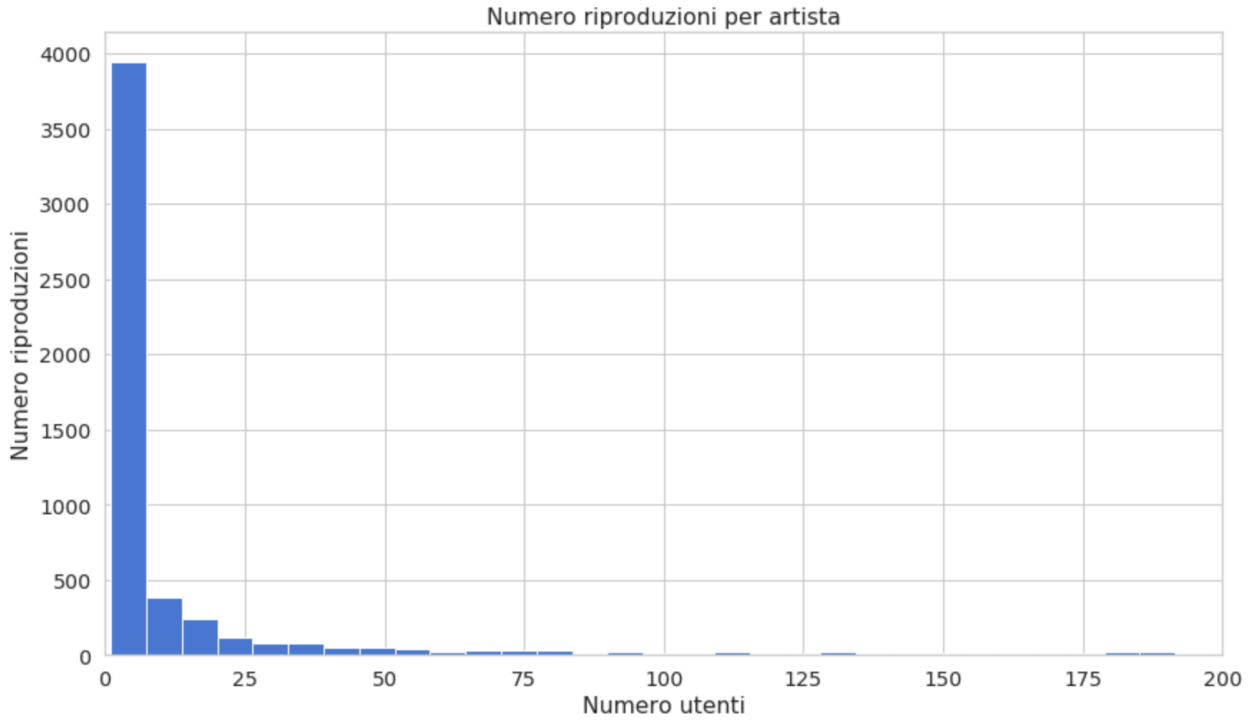
```
[ ]  artist_rank = ap.groupby(['name']) \
       .agg({'userID' : 'count', 'numeroRiproduzioni' : 'sum'}) \
       .rename(columns={"userID" : 'totaleUtenti', "numeroRiproduzioni" : "totaleArtistiRiprodotti"}) \
       .sort_values(['totaleArtistiRiprodotti'], ascending=False)

     artist_rank['mediaRiproduzioniUtente'] = artist_rank['totaleArtistiRiprodotti'] / artist_rank['totaleUtenti']
```

A questo punto è possibile andare ad unire i dati con il dataframe precedente, che è stato pre-processato:

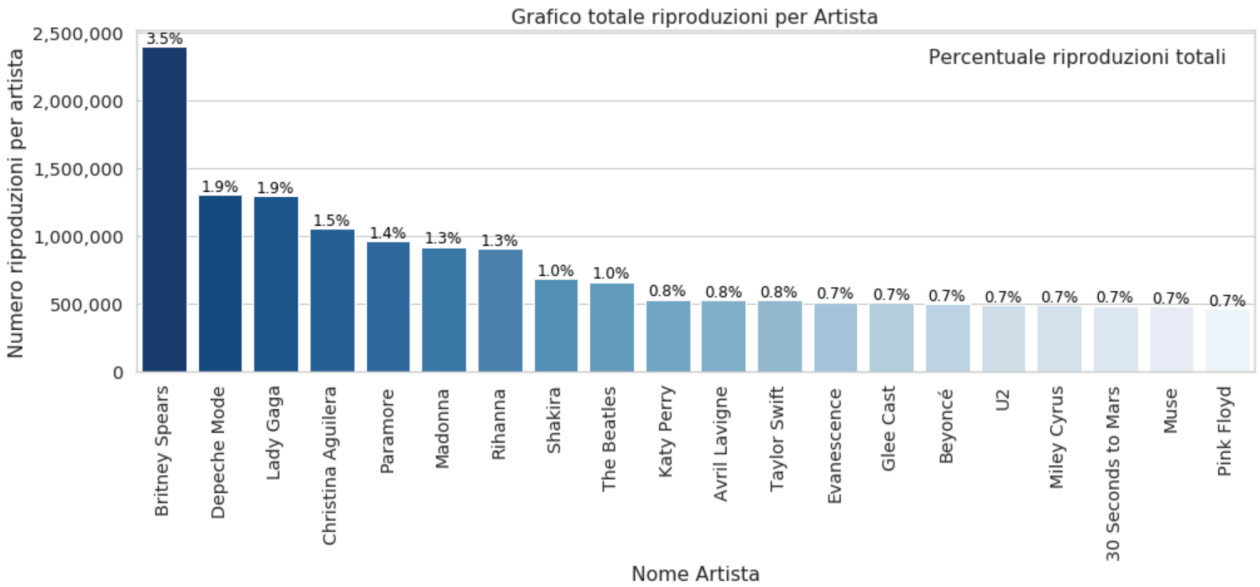
```
[ ]  ap = ap.join(artist_rank, on="name", how="inner") \
       .sort_values(['numeroRiproduzioni'], ascending=False)
```

5 - Esplorazione dei dati

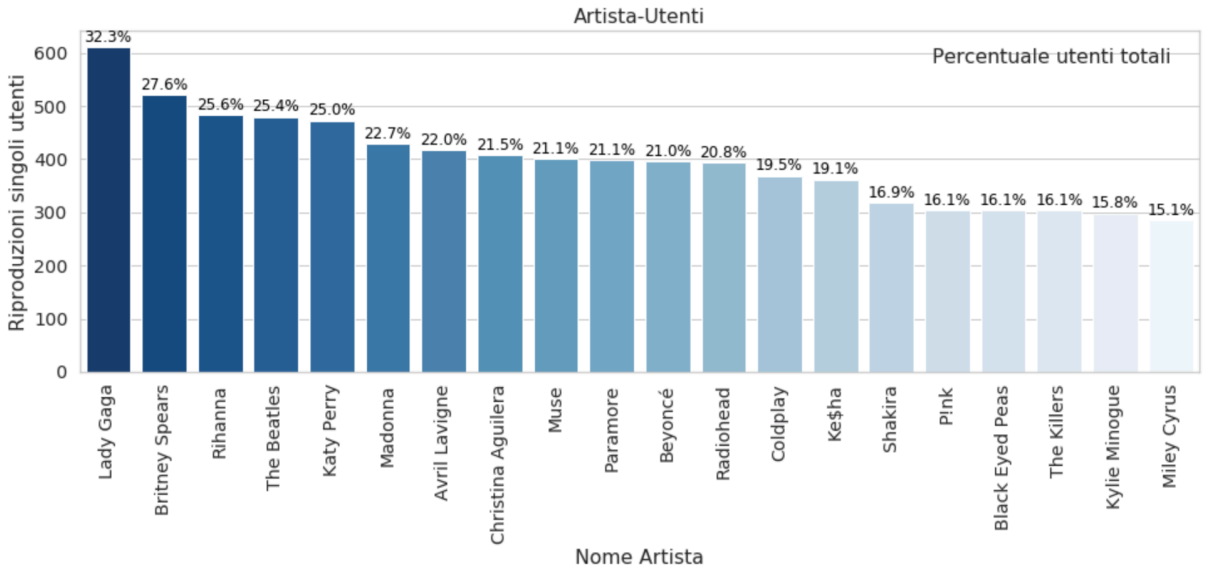
Il grafico seguente mostra quanto gli artisti siano stati riprodotti dagli utenti.



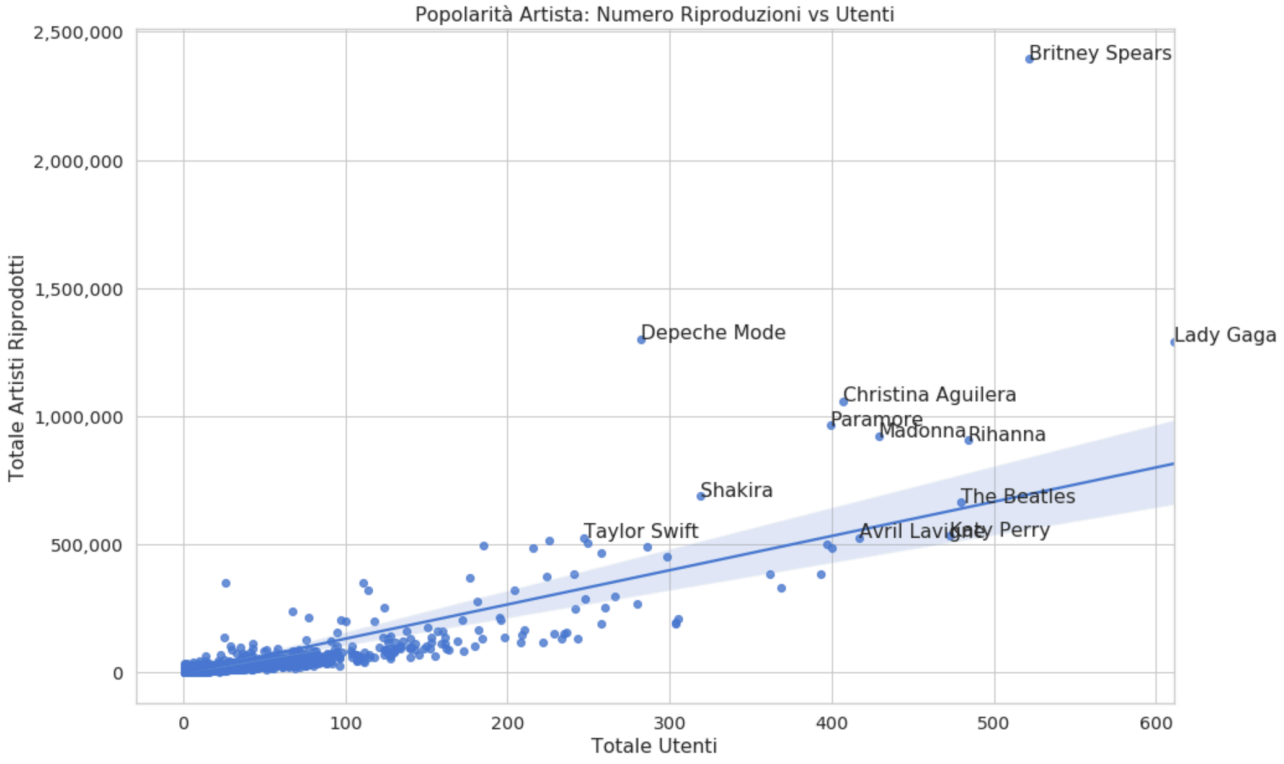
Di seguito invece i nomi degli artisti, estratti dal dataset last.fm, che sono stati riprodotti maggiormente:



Quanti degli utenti riproducono l'artista:



Ecco un altro grafico della popolarità degli artisti:



Gli artisti più popolari, come si evince dai grafici, ovviamente ricevono il maggior numero di riproduzioni.

6 - Sistemi di raccomandazione

I RS sono usati praticamente ovunque ci sia una vasta gamma di elementi tra cui scegliere.

Funzionano fornendo suggerimenti che potrebbero aiutare a fare scelte migliori/più veloci.

I RS sfruttano modelli di comportamento condivisi da utenti.
Ad esempio tu e i tuoi amici potreste avere gusti simili per alcune cose.

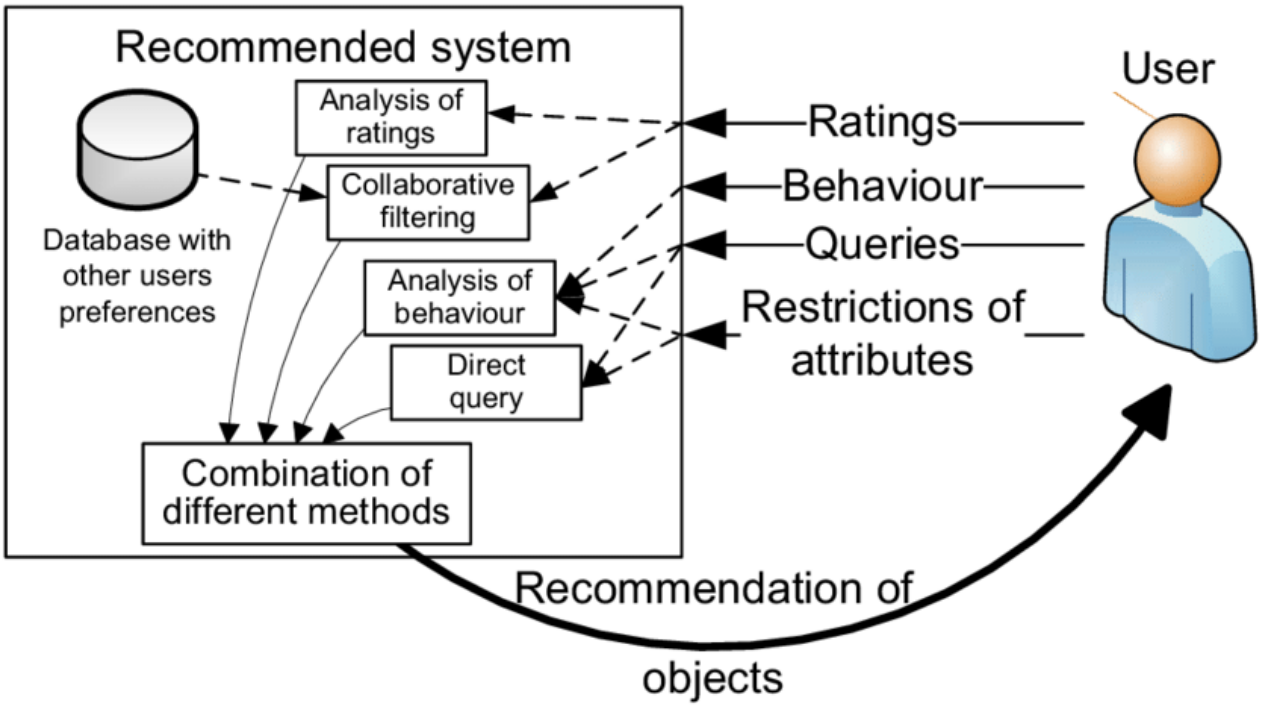
Un RS prova a trovare “amici” all’interno di altri utenti e consiglia cose che non hai provato.

Le due tipologie di RS più usati sono:

1. **CONTENT-BASED**
2. **COLLABORATIVE FILTERING (CF)**

Il CF produce raccomandazioni basate su conoscenza degli utenti rispetto agli item, ovvero utilizza la “saggezza della folla” per raccomandare gli item.

Al contrario, i RS basati sul contenuto si concentrano sulle proprietà degli item e forniscono consigli basati sulla loro somiglianza.



ARCHITETTURA RS, FONTE: WWW.RESEARCHGATE.NET

7 - Collaborative filtering

Il CF è il punto di forza dei RS.
La peculiarità principale dell'algoritmo è quella di essere in grado di eseguire l'apprendimento delle funzionalità, che gli consente di iniziare a imparare quali funzioni utilizzare.

Può essere suddiviso in due categorie:

1. ***MEMORY-BASED CF:*** suddivisi a loro volta in due sezioni:

    1.1 USER-ITEM FILTERING: "anche agli utenti simili a te è piaciuto l'item…"
    1.2 ITEM-ITEM FILTERING: "agli utenti a cui è piaciuto questo item è piaciuto anche…"

Entrambi i metodi richiedono una matrice user-item che contiene la valutazione dell'utente ***u*** per l'item ***i***.

Da questo si può calcolare la matrice di similarità.
I valori di similarità nel CF item-item vengono calcolati prendendo in considerazione tutti gli utenti che hanno valutato una coppia di item.
Per il CF user-item, i valori di similarità sono calcolati osservando tutti gli elementi classificati da una coppia di utenti.

2. MODEL-BASED CF: I CF model-based si basano sulla *MATRIX-FACTORIZATION* (MF).

I metodi MF sono usati come metodo di apprendimento senza supervisione per la decomposizione della latenza e la riduzione della dimensionalità.

Lo scopo dei metodi per la riduzione della dimensionalità, è quello di eseguire un mapping dallo spazio iniziale $\mathfrak{R}^d$ verso un nuovo spazio con meno dimensioni $\mathfrak{R}^k$.
Può essere intesa come una compressione con perdita di informazione, il cui obiettivo è quello di scartare le informazioni non rilevanti o meno rilevanti per il problema di interesse.
Grazie a questi metodi si alleviano i problemi collegati alla **curse of dimensionality** in quanto operare in spazi ad elevata dimensionalità, a causa del fatto che i pattern sono troppo sparsi richiede molti dati per l'addestramento.
Invece, operare in spazi a dimensione inferiore permette di migliorare le prestazioni.
Ridurre la dimensionalità non significa mantenere delle dimensioni e scartarne delle altre, ma significa combinare in maniera opportuna le dimensioni che esistono.

Nel corso di Machine Learning abbiamo approfondito tre particolari tecniche che ci permettono di fare ciò:

1. **Principal Component Analysis (PCA):** prevede una trasformazione non-supervisionata ed esegue un mapping lineare con l'obiettivo di preservare al massimo l'informazione dei pattern.
2. **Linear Discriminant Analysis (LDA):** dove il mapping ancora una volta è lineare ma supervisionato. Mentre PCA privilegia le dimensioni che rappresentano al meglio i pattern, LDA privilegia le dimensioni che discriminano al meglio i pattern nel Training Set[4].
3. **t-distributed Stochastic Neighbor Embedding (t-SNE):** la quale è una trasformazione non lineare e non supervisionata ideata per ridurre la dimensionalità a 2 o 3 dimensioni per poter visualizzare dati multidimensionali.

Inoltre abbiamo citato anche la I**ndependent Component Analysis**, **Kernel PCA** e la **Local Linear Embedding**.

Gestiscono in maniera più adeguate dei CF MEMORY_BASED, i problemi di scalabilità e di data sparsity.

L'obbiettivo della MF è apprendere le preferenze dell'utente corrente e gli attributi degli item da classificare.

La MF ristruttura la matrice user-item in una matrice di basso livello.
Si può rappresentare moltiplicando due matrici con rango basso, in cui le righe contengono un vettore di variabili latenti.
In questo modo si prevedono le voci mancanti nella matrice originale delle valutazioni.

[4] In apprendimento automatico un **training set** (o insieme di addestramento) è un insieme di dati che vengono utilizzati per addestrare un sistema supervisionato

8 - Singular Value Decomposition

La decomposizione ai Valori Singolari (SVD) è un'importante fattorizzazione per matrici a valori reali o complessi che si avvale dei concetti di autovalori e autovettori.

Ogni matrice $M \in C^{mxn}$ può essere fattorizzata in: $M = U\Gamma V*$
Dove:
- U è una matrice mxm unitaria
- Γ è una matrice mxn diagonale rettangolare con soli elementi reali non negativi.
- $V*$ è la trasposta coniugata di una matrice nxn unitaria V.

Il CF può essere formulato approssimando una matrice X usando la **DECOMPOSIZIONE A VALORI SINGOLARI (SVD).**

Il team vincitore alla "Netflix Prize Competition"[5] ha usato una SVD matrix factorization per vincere.
SVD può essere espresso come:

$$X = (U \cdot S \cdot V)^T$$

Data una matrice X [m x n] :
- U è una matrice ortogonale [m x n]
- S è una matrice diagonale con numeri reali non-negativi sulla diagonale [r x r]
- U^T è una matrice ortogonale [r x n]

Dove U rappresenta il vettore delle features degli utenti, V rappresenta il vettore delle features degli item e gli elementi sulla diagonale di S sono conosciuti come SINGULAR VALUES.

Si può fare una predizione tramite il prodotto di U, S, V^T.

[5] Il Premio Netflix è stato un concorso aperto per il miglior algoritmo di filtro collaborativo per prevedere le valutazioni degli utenti per i film, in base alle valutazioni precedenti.

9 - Consigliare artisti musicali

Anche se la maggior parte dei tutorial, visti su internet si concentra sugli approcci basati sulla memoria, questi non sono usati nella pratica, poiché producono buoni risultati ma non si adattano bene al problema del **COLD-START**.

D'altra parte, l'applicazione dei SVD richiede la fattorizzazione della matrice user-item e ciò può essere costoso quando la matrice è molto sparsa (poiché mancano molte valutazioni user-item).

Approcci più recenti si sono concentrati sulla previsione delle valutazioni riducendo al minimo l'errore quadratico regolarizzato rispetto ad una matrice user-feature P e la latenza per l'item feature matrix Q**:**

$$\min_{Q*,P*} \sum_{(u,i)\in K} (r_{ui} - P_u^T Q_i)^2 + \lambda(||Q_i||^2 + ||P_u||^2)$$

Dove K è un insieme di coppie (u, i), $r(u, i)$ è la valutazione dell'utente u per l'item i e λ è un termine di regolarizzazione per evitare **l'OVERFITTING**.

L'addestramento del modello consiste nel ridurre al minimo l'errore al quadrato regolarizzato.
Dopo aver ottenuto una stima di P e Q, si possono prevedere valutazioni sconosciute facendo il prodotto delle funzionalità latenti per gli utenti e gli item.

Possiamo applicare lo **STOCHASTIC GRADIENT DESCENT(SGD)** o l'**ALTERNATING LEAST SQUARES(ALS)** per minimizzare le perdite.

Entrambi i metodi possono essere utilizzati per aggiornare progressivamente il nostro modello con l'arrivo di nuove valutazioni.
Si è deciso di applicare lo SGD poiché sembra essere generalmente più accurato e veloce di ALS (tranne in situazioni con dati altamente scarsi e impliciti).

Inoltre SGD è ampiamente usato per la formazione dei reti neurali profonde.

Nello specifico, come appreso durante il corso di Machine Learning, nel Gradient Descent gli esempi di addestramento vengono fatti passare attraverso la rete uno per volta.

L'addestramento analizza l'intero batch, cioè l'insieme di tutte le istanze.
Questo processo viene chiamato epoca.

Per evitare l'underfitting dei parametri si considerano più epoche, fino ad un ben preciso criterio di terminazione.

Di conseguenza la discesa del gradiente è un processo iterativo che deve essere eseguito molto volto e il suo calcolo richiederà molte risorse sull'intero training set.

Nello SGD o Gradient Descent iterativo o ancora Gradient Descent online, non si sommano gli aggiornamenti dei pesi per tutte le istanze, ma si aggiornano dopo ogni istanza.

Quindi è stocastico perché si approssima il gradiente reale con approssimazioni successive, considerando un solo training alla volta.

Per cui si cerca di convergere con approssimazioni successive dell'errore e per ipotesi si converge con un numero di iterazioni che è nettamente inferiore rispetto alla grandezza del training set.

Inoltre si possono aggiornare i pesi in maniera più puntuale e quindi la rete si può monitorare in maniere opportuna.

Per evitare di convergere dopo molto tempo si possono considerare piccoli gruppi di istanze alla volta chiamate **minibatch**.

Il numero di minibatch corrisponde al numero di iterazioni per un epoca.

Se ad esempio si hanno 2000 istanze nel training set, e minibatch di 500 istanze allora saranno necessarie 4 iterazioni per completare un processo.

Ovviamente il minibatch introduce della varianza nella variazione dei parametri durante l'addestramento poiché per limitare gli errori si tendono a fare molti più passi, con un **Learning rate** più basso.

Inoltre il Learning rate può essere reso dinamico, in funzione di quanto reputo sia distante dal punto ottimale.

All'inizio sarà elevato così ci potremmo muovere nella rete più velocemente, mentre quando i parametri tendono a quelli ottimali andrò a ridurre il rate in questo modo:

$$-\alpha^t \Delta_w l^t$$

Per adattare il rate si possono utilizzare l'**inverse time decay** o l'**exponential decay**.

Per approfondimenti si vadano le slides del corso di Machine Learning.

Inoltre con la tecnica del **Momentum** possiamo pensare di creare una “inerzia” nella ricerca successiva, memorizzando prima gli ultimi aggiornamenti e favorendo la ricerca verso la direzione più verosimile in base alle ultime osservazioni:

$$v_w^t = \beta v_w^{t-1} + \varepsilon \Delta_w l^t$$

$$w^{t+1} \leftarrow w^t - \alpha v_w^t$$

10 - Pre-processamento dei dati

Per applicare l'algoritmo di CF , dobbiamo trasformare il nostro set di dati in una matrice utente-artista.

Prima di tutto andiamo a ridimensionare e quindi filtrare i dati:

```
pc = ap.numeroRiproduzioni
play_count_scaled = (pc - pc.min()) / (pc.max() - pc.min())

ap = ap.assign(playCountScaled=play_count_scaled)
```

Ciò riduce il conteggio all'intervallo [0 -1] e aggiunge una nuova colonna al nostro frame di dati (come viene fatto anche nel **Boosting** per l'aggiunta del peso).

A questo punto costruiamo il nostro frame di dati "valutazioni":

```
ratings_df = ap.pivot(
    index='userID',
    columns='artistID',
    values='playCountScaled'
)
```

Un metodo di Pandas ci consente di creare un frame di dati indice/colonna e di riempire gli spazi mancanti con 0.
Da ciò che segue si può notare quanto è scarso il frame di dati:

```
[ ]  sparsity = float(len(ratings.nonzero()[0]))
     sparsity /= (ratings.shape[0] * ratings.shape[1])
     sparsity *= 100
     print('{:.2f}%'.format(sparsity))
```

```
0.28%
```

Il dataset per ora è ancora troppo sparso
Per questo suddividiamo i nostri dati in **training set** da addestrare e **validation set**:

```
[ ]  train, val = train_test_split(ratings)
```

Ed inoltre definiamo la funzione **train_test_split** in un modo leggermente diverso:

```
[ ]  MIN_USER_RATINGS = 35
     DELETE_RATING_COUNT = 15

     def train_test_split(ratings):

         validation = np.zeros(ratings.shape)
         train = ratings.copy()

         for user in np.arange(ratings.shape[0]):
             if len(ratings[user,:].nonzero()[0]) >= MIN_USER_RATINGS:
                 val_ratings = np.random.choice(
                     ratings[user, :].nonzero()[0],
                     size=DELETE_RATING_COUNT,
                     replace=False
                 )
                 train[user, val_ratings] = 0
                 validation[user, val_ratings] = ratings[user, val_ratings]
         return train, validation
```

Infine, rimuoviamo alcuni conteggi sostituendoli con zeri.

11 - Valutazione dell'errore

Una delle metriche più popolari utilizzate per valutare l'accuratezza dei sistemi di raccomandazione è il **ROOT MEAN SQUARED ERROR (RMSE)**, definito come:

$$RMSE = \sqrt{\frac{1}{N}\sum (y_i - \hat{y}_i)^2}$$

Dove y_i è il valore reale per l'item i, $\hat{y}_i$ è il valore predetto e N è la dimensione del set di training.
In Python l'RMSE si calcola nel seguente modo:

```
[ ]  def rmse(prediction, ground_truth):
         prediction = prediction[ground_truth.nonzero()].flatten()
         ground_truth = ground_truth[ground_truth.nonzero()].flatten()
         return sqrt(mean_squared_error(prediction, ground_truth))
```

12 - Training

Addestriamo il Reccomender System tramite lo SGD:

```
    def fit(self, X_train, X_val):
      m, n = X_train.shape

      self.P = 3 * np.random.rand(self.n_latent_features, m)
      self.Q = 3 * np.random.rand(self.n_latent_features, n)

      self.train_error = []
      self.val_error = []

      users, items = X_train.nonzero()

      for epoch in range(self.n_epochs):
          for u, i in zip(users, items):
              error = X_train[u, i] - self.predictions(self.P[:,u], self.Q[:,i])
              self.P[:, u] += self.learning_rate * (error * self.Q[:, i] - self.lmbda * self.P[:, u])
              self.Q[:, i] += self.learning_rate * (error * self.P[:, u] - self.lmbda * self.Q[:, i])

          train_rmse = rmse(self.predictions(self.P, self.Q), X_train)
          val_rmse = rmse(self.predictions(self.P, self.Q), X_val)
          self.train_error.append(train_rmse)
          self.val_error.append(val_rmse)

      return self

    def predict(self, X_train, user_index):
      y_hat = self.predictions(self.P, self.Q)
      predictions_index = np.where(X_train[user_index, :] == 0)[0]
      return y_hat[user_index, predictions_index].flatten()
```

Sono presenti due matrici una per gli utenti e una per la valutazione.
Per ogni coppia utente e item, addestriamo e calcoliamo l'errore (usando una semplice differenza tra output predetto e output atteso).
Quindi aggiorniamo P e Q usando il gradient descent.

Dopo ogni epoca di addestramento, calcoliamo gli errori di addestramento e di convalida e memorizziamo i loro valori per analizzarli in seguito.

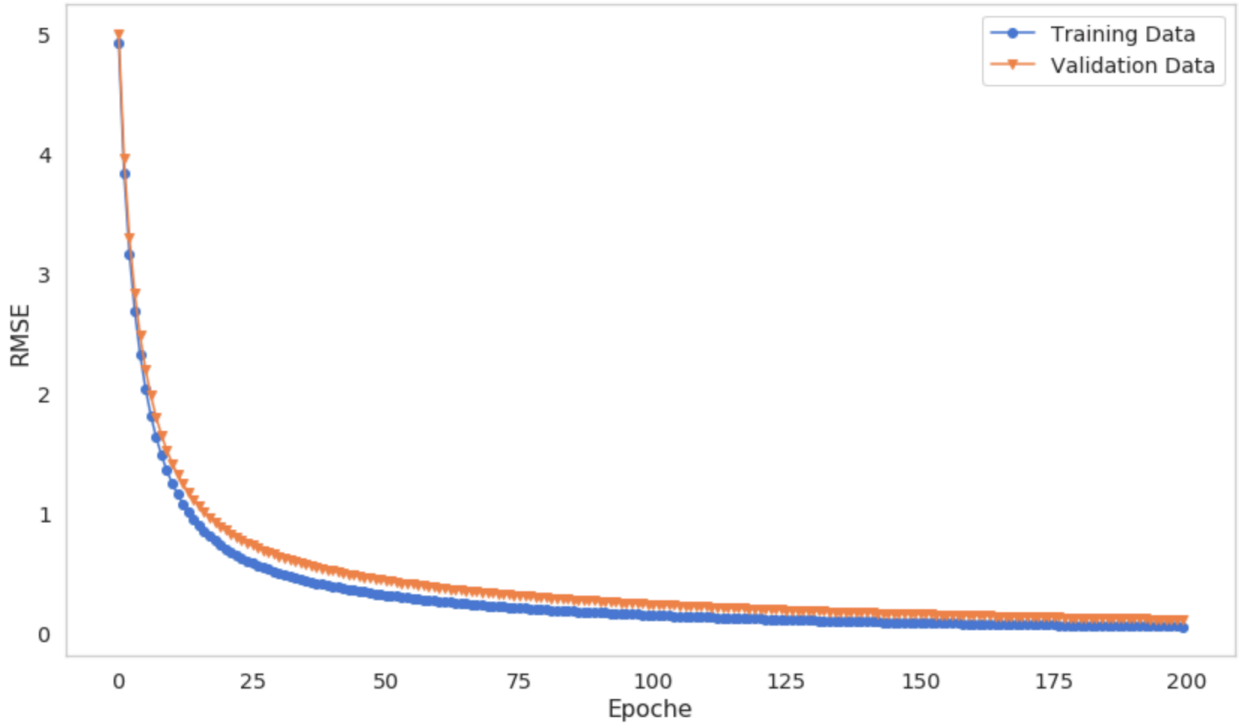
Ecco di seguito l'implementazione del metodo delle previsioni:

```
def predictions(self, P, Q):
  return np.dot(P.T, Q)
```

Come discusso prima, otteniamo previsioni tramite il prodotto della P^T e Q.

13 - Test

Vediamo come è andato l'addestramento esaminando il training e l'RMSE:



Il modello sembra "allenarsi" gradualmente ed entrambi gli errori diminuiscono con l'aumentare del numero di epoche.
Usando ancora più tempo il modello potrebbe migliorare ancora.

Aumentando infatti il parametro relativo al tempo si sono ottenuti risultati sperimentali migliori in termini di accuratezza.

Ovviamente ne risente l'efficienza, poiché bisogna attendere molto più tempo prima che avvenga l'addestramento completo.

14 - Risultati finali

Dopo l'addestramento il reccomender è pronto a consigliare ad un utente un artista da ascoltare.
Di seguito l'implementazione del metodo di predict:

```
def predict(self, X_train, user_index):
  y_hat = self.predictions(self.P, self.Q)
  predictions_index = np.where(X_train[user_index, :] == 0)[0]
  return y_hat[user_index, predictions_index].flatten()
```

Otteniamo quindi una matrice che ci restituisce le previsioni.

In secondo luogo, prendiamo gli indici di tutti gli elementi sconosciuti e restituiamo solo questi come previsioni.

Vediamo quali sono le attuali preferenze di un utente specifico:

```
[ ]  user_id = 1236
     user_index = ratings_df.index.get_loc(user_id)
     predictions_index = np.where(train[user_index, :] == 0)[0]

     rating_predictions = recommender.predict(train, user_index)
```

```
[ ]  def create_artist_ratings(artists_df, artists_index, ratings, n=10):
       artist_ids = ratings_df.columns[artists_index]
       artist_ratings = pd.DataFrame(data=dict(artistId=artist_ids, rating=ratings))
       top_n_artists = artist_ratings.sort_values("rating", ascending=False).head(n)

       artist_recommendations = artists_df[artists_df.id.isin(top_n_artists.artistId)].reset_index(
       artist_recommendations['rating'] = pd.Series(top_n_artists.rating.values)
       return artist_recommendations.sort_values("rating", ascending=False)
```

```
[ ]  existing_ratings_index = np.where(train[user_index, :] > 0)[0]
     existing_ratings = train[user_index, existing_ratings_index]

     create_artist_ratings(artists, existing_ratings_index, existing_ratings)
```

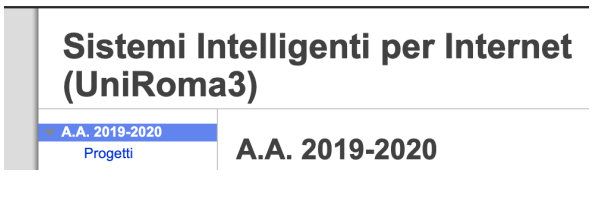
	id	name	rating
0	7	Marilyn Manson	0.196486
1	472	3 Doors Down	0.043204
2	614	Pearl Jam	0.042016
3	923	Children of Bodom	0.025657
4	978	Disturbed	0.021690
5	1104	Rammstein	0.021562
6	1118	A Perfect Circle	0.020879
7	2172	Gojira	0.017051
8	6257	Rob Zombie	0.016280
9	6618	D12	0.010990

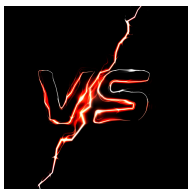
Vediamo infine quali sono gli artisti suggeriti dal sistema per l'utente specifico:

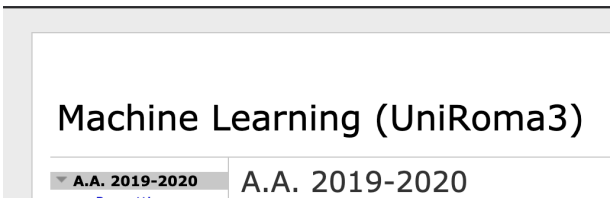
```
[ ]  create_artist_ratings(artists, predictions_index, rating_predictions)
```

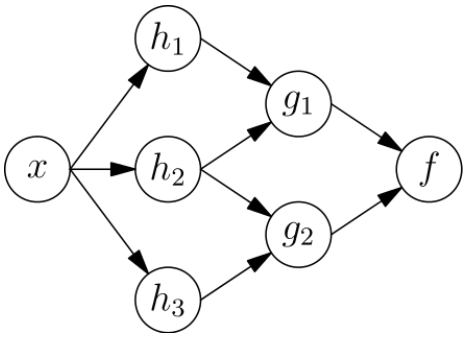
	id	name	rating
0	1318	Sander van Doorn	0.559202
1	3167	Jacob Miller	0.552893
2	7627	Celtas Cortos	0.552142
3	10995	Camisa de Vênus	0.546361
4	12186	Ella Fitzgerald & Louis Armstrong	0.541477
5	12911	The Answer	0.537367
6	14245	Anjelika Akbar	0.536756
7	14940	Medications	0.536486
8	15792	Lützenkirchen	0.535515
9	18559	Archie Star	0.535147

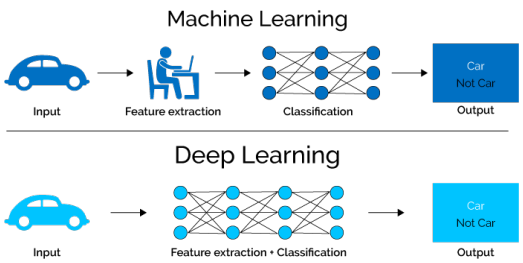
Considerazioni finali e confronto











Sono state applicate (quasi) tutte le nozioni apprese durante i corsi.

In particolare per quanto riguarda il corso di Sistemi Intelligenti per Internet l'attenzione è stata posta sul Reccomender System, uno degli argomenti cardine.

Il sistema è stato addestrato tramite i metodi presentati durante il corso di Machine Learning, in particolare mi sono focalizzato sullo SGD, argomento rilevante del corso e approfondito nella parte relativa al Deep Learning.

What's new?

Modificando opportunamente i parametri, presenti in colab, ed in particolare nella parte relativa appunto allo SGD, i tempi di addestramento possono risultare molto più veloci rendendo quindi tutta la computazione più efficiente.

Inoltre mi sono focalizzato più nello specifico nella misura dell'errore RMSE ed in particolare sulla decomposizione a valori singolari, essendo questa ultima più recentemente studiata.

Infine ho studiato in maniera più approfondita il dataset *last.fm* che ritengo estremamente valido ed utile per questo tipo di lavori.

Dopo aver affrontato, a livello teorico e più nel dettaglio la tecnica dello SGD, è risultato più semplice capire il funzionamento del sistema, che in precedenza lasciava punti oscuri su più fronti.
Di fatti la tecnica del momentum è risultata essere semplice ed intuitiva.

Si sono approfondite inoltre le motivazioni che non hanno portato ad utilizzare dei metodi per la riduzione delle dimensionalità e quindi ad adottare un model-based CF.

Ritengo che il progetto sia stato stimolante, in quanto su uno scenario reale, mi ha mostrato praticamente, come le nozioni teoriche possano davvero funzionare!

Sarebbe divertente e ancora più stimolante, capire quello che c'è dietro ai grandi colossi, che della previsione e dei suggerimenti all'utente campano quotidianamente.

Per il codice si può visitare la mia pagina GitHub a *questo indirizzo*.
Il codice poi è da far girare su Google Colaboratory.

Riferimenti

- Slides dei corsi di *Machine Learning* e *Sistemi Intelligenti per Internet*
- *https://it.wikipedia.org/wiki/Discesa_stocastica_del_gradiente*
- *https://www.retineuraliartificiali.net/deep-learning/google-colaboratory/*
- *https://www.learnpython.org/en/Pandas_Basics*
- *http://disi.unitn.it/~passerini/teaching/2017-2018/informatica/slides/23_pandas/pandas.html*
- *https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm*
- *https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31*
- *https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e*
- *https://www.last.fm*
- *http://www.linux-magazine.it/last-fm-tutta-unaltra-musica/*