Overview of Netflix Prize challenge

Enzo Calabrese

August 2025

Contents

1 Introduzione

Il Netflix Prize è una competizione creata proprio da Netflix nel 2006, per lo sviluppo di un nuovo sistema di raccomandazione con prestazioni superiori rispetto al loro algoritmo proprietario "Cinematch". E' stato rilasciato un dataset con ben 100 milioni di rating, espressi da 480189 utenti su 17770 film, divisi in due differenti dataset:

- Training set: contiene 100,480,507 rating, con 1,408,395 di essi che facevano parte del cosiddetto "probe set", il quale serviva per la validazione dei modelli;
- Qualifying set: contiene 2,817,131 ratings, e fungeva da test set. Questo dataset è stato a sua volta diviso in due sottoinsiemi: il test set ed il quiz set. Il test set conteneva 1,408,789 ed è stato utilizzato da Netflix alla fine della competizione per decretare il vincitore; il quiz set, con 1,408,342 rating, era invece usato per calcolare gli score che comparivano nella classifica pubblica, sul loro sito. Ai partecipanti non venivano offerte le vere etichette corrispondenti a questi set, per evitare che essi barassero, includendoli nel training set.

Tutti i subset utilzzati per validazione (probe) e per testing (test/quiz) erano costruiti appositamente per condividere le stesse proprietà statistiche (es. distribuzione dei rating, numero di film/utenti con pochi rating e film/utenti con molti rating, ecc.). Netflix sosteneva che il loro algoritmo Cinematch totalizzasse 0.9514 di RMSE sul quiz set; per vincere la sfida, era necessario realizzare un modello che battesse di almeno il 10% Cinematch, in termini di RMSE; in altre parole, doveva totalizzare 0.8562 RMSE sul quiz set, ed ovviamente essere il primo in classifica. Il premio finale, pari ad 1 milione di dollari, è stato vinto dal modello "BellKor's Pragmatic Chaos", creato dai team "BellKor" (Robert Bell e Chris Volinsky da AT&T, Yehuda Koren da Yahoo), "BigChaos" (Andreas Töscher e Michael Jahrer) e "Pragmatic Theory" (Martin Piotte e Martin Chabbert). Questo modello è riuscito a migliorare lo score di Cinematch del 10.09% "The Ensemble", il secondo in classifica, ha registrato un miglioramento del 10.10% sul quiz set, ma BellKor ha performato meglio nel test set, da cui la vittoria.

1.1 Il mio scopo per questo progetto

Per prima cosa, voglio dare al lettore una panoramica completa di questa competizione, e, in generale, delle nozioni di base riguardanti i sistemi di raccomandazione collaborative filtering. Si partirà dal principio, dai modelli più basilari, come ad esempio i modelli collaborative filtering item-item attraverso similarità del coseno; poi si passerà ai modelli basati su matrix-factorization (una tecnica simile alla SVD), che sono stati la base dei migliori modelli usciti fuori dal Netflix Prize.

Il mio obiettivo principale è però quello di esplorare gli avanzamenti nel campo a partire dalla fine della challenge fino al giorno d'oggi, per capire se, al contrario di 16 anni fa, sia possibile costruire in maniera relativamente semplice un modello in grado di batterla. Questo obiettivo è molto ambizioso, essendo che si tratta di quella che potrebbe essere chiamata una "blind run" (la mia conoscenza attuale si esaurisce agli argomenti trattati nel corso), quindi quasi sicuramente non sarà raggiunto; sono però sicuro che, lavorando per raggiungere l'obiettivo in questione, la mia comprensione dell'ambito dei sistemi di raccomandazione non potrà fare altro che aumentare.

1.2 Vincoli

- Il dataset su cui lavorerò deve essere il dataset originale fornito da Netflix. Potrò arricchirlo utilizzando ogni metodo a mia disposizione, anche integrando i dati di altri dataset, come IMDB e TMDB, ma il test set finale dovrà comunque essere il probe set o una sua parte;
- Non potrò usare soltanto implementazioni già esistenti degli algoritmi in questione, limitandomi ad una banale sequela di chiamate a funzioni di libreria; dovrò sviluppare almeno gli algoritmi principali (CF e Matrix Factorization) con le mie mani, appoggiandomi soltanto alle librerie base di Python come NumPy, oppure PyTorch;

- Devo riuscire a portare a termine il mio compito facendo unicamente affidamento alla mia macchina locale, con Intel Core i7 6700 Skylake, 16 GB RAM DDR4, NVIDIA GTX 1070.

1.3 Perchè uso il probe set come test set?

Come già detto, il qualifying set è stato rilasciato senza i rating corrispondenti, e questi ultimi non sono stati rilasciati neanche dopo la fine della challenge. Per questo motivo, dovrò necessariamente utilizzare come test set proprio il probe set. Questa è circa la configurazione utilizzata dai team partecipanti alla challenge per i loro test iniziali e validazione degli iperparametri, per cui i punteggi dovrebbero essere simili a quelli ottenuti da loro; gli iperparametri da me utilizzati saranno, quando possibile, gli stessi utilizzati dai team e descritti nei paper corrispondenti.

2 Panoramica dei metodi principali per raccomandazione tramite CF

Il dataset del Netflix Prize è disponibile su Kaggle a questo link. Lo zip contiene:

- Il training set, diviso in 4 file txt chiamati "combined_data.txt". L'archivio tar originale condiviso da Netflix ha 17700 file txt, uno per ogni film.
- Il qualifying set.
- Il probe set.
- Un file .csv che contiene, per ogni id del film, il suo titolo.

Tutti i file .txt dei rating hanno una struttura del genere:

- La prima riga è l'id del film seguito dai due punti;
- Le altre righe rappresentano i rating degli utenti, con id utente, punteggio (numero intero tra 1-5) e data del rating nel formato YYYY-MM-DD.

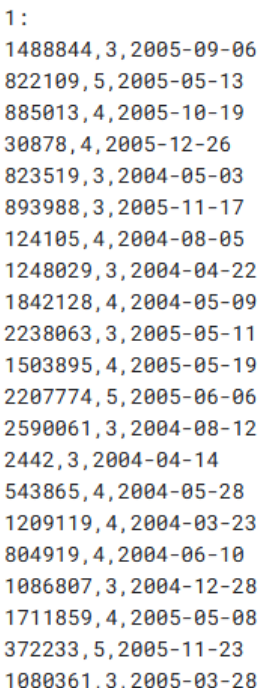

Figure 1: La struttura di ogni txt.

2.1 Estrazione dei dati di addestramento dai file txt

Ho scritto un piccolo script Python (extract_data_mapping_sparse.py) per leggere tutte le righe dei file txt del training set e creare la matrice di utilità con scipy.sparse. Inserirò gli id dei film, gli id degli utenti e i rating in 3 liste Python.

Mapping degli id utente. Per permettere la costruzione della matrice di utilità, ho creato una mappa degli id utente: essi non sono di default progressivi, da 1 a 480189. Devo associare ogni utente a una riga della matrice, e per questo motivo ho mappato gli id utente con id progressivi da 0 a 480188.

Mapping degli id film. Stessa cosa con gli id dei film, ma questi sono effettivamente progressivi, da 1 a 17770. Per associarli alle colonne della matrice, li ho semplicemente decrementati di 1.

2.2 Matrici sparse

Nota: poiché il dataset contiene 480189 utenti e 17770 film, la matrice di utilità risultante avrebbe circa 8 miliardi di elementi (la maggioranza assoluta di questi pari a zero); dato che gli int di NumPy sono 4 byte, la matrice occuperebbe circa 30 GB di RAM. Per questo motivo, abbiamo bisogno di una rappresentazione a matrice sparsa.
Ho convertito le tre liste Python in array NumPy, per risparmiare memoria (perché le liste Python portano alcuni metadati per ogni elemento; gli array NumPy contengono solo valori int a 4 byte e puntatori). Infine, ho convertito i tre array NumPy in una matrice sparsa CSR di SciPy. Gli id utente rappresentano le righe delle matrici, gli id film rappresentano le colonne delle matrici e i rating sono contenuti in ogni cella [id utente][id film].
Ho scelto di creare unicamente la rappresentazione CSR perchè nella maggior parte dei casi avrò bisogno di indicizzarla per riga (utente); nei rari casi in cui è necessario avere una rappresentazione CSR, SciPy offre una semplicissima API in grado di effettuare la conversione.

2.3 Collaborative filtering basato su similarità

Uno degli algoritmi più semplici per la raccomandazione è il **collaborative filtering**. Il rating di un utente per un item è predetta basandosi su:

- Rating di utenti simili per lo stesso item;
- Rating dello stesso utente per item simili.

Per predire il rating di un utente per un item, l'algoritmo di collaborative filtering calcola le similarità (spesso la similarità del coseno) tra utenti/item. Vengono selezionati i k utenti/item più simili, e viene calcolata una media pesata (i pesi sono le similarità) tra tutte queste valutazioni, o di utenti simili date all'item in questione, oppure dello stesso utente agli item più simili. Il primo tipo di collaborative filtering è chiamato **user-user**, e il secondo tipo è chiamato **item-item**.

Collaborative filtering user-user. Dato l'utente i e l'item j, il rating r_{ij} dell'utente i per l'item j è calcolata con la seguente espressione:

$$r_{ij} = \frac{\sum_k s_{ik} r_{kj}}{\sum_k s_{ik}}$$

Collaborative filtering item-item. Dato l'utente i e l'item j, il rating r_{ij} dell'utente i per l'item j è calcolata con la seguente espressione:

$$r_{ij} = \frac{\sum_k s_{jk} r_{ik}}{\sum_k s_{jk}}$$

2.3.1 Item-item collaborative filtering

Ho implementato una funzione Python (in src_naive_cf/naive_cf.py) per predire i rating del probe set con il collaborative filtering item-item. Nel nostro scenario è meglio un approccio item-item perché il numero di utenti è troppo alto. Per predire un singolo rating, dovremmo calcolare ben 480188 similarità. Nella mia implementazione uso una matrice di similarità 17770×17770 precalcolata, per aumentare la velocità di predizione. Se dovessi calcolare la matrice di similarità tra utenti, otterrei una matrice 480189×480189, che non entra in memoria. Anche senza queste matrici la situazione rimarrebbe la stessa: il calcolo della similarità è molto più lento con un filtering collaborativo user-user.

Risultati. Ho predetto tutti i rating del probe set con il mio algoritmo di collaborative filtering item-item (src_naive_cf/naive_cf.py), per diversi valori di k.

Table 1: RMSE del collaborative filtering item-item con diversi valori di k.

k	RMSE
2	1.1273142661038915
5	1.0446833312162191
10	1.025567438348012
20	1.0243055494191222
50	1.0334593290470646
100	1.0401581919526133

Confrontiamo con le predizioni baseline (src_naive_cf/baseline.py) basate sul rating medio globale, sulla media dei rating dell'utente (ogni rating predetto dell'utente x corrisponde alla media di x) e sulla media dei rating del film (ogni rating predetto per il film y corrisponde allo score medio di y).

Table 2: RMSE delle predizioni baseline

Tipo di media	RMSE
Media globale	1.1296234689653422
Media utente	1.0687865426639003
Media film	1.0528157616564027

Ovviamente, il collaborative filtering item-item supera le predizioni baseline: ma è troppo alto rispetto al nostro obiettivo. Per esempio, il Cinematch di Netflix ottiene 0.9514 anche sul probe set.

2.3.2 Normalizzazione dei rating

Questo algoritmo di collaborative filtering non considera che ogni utente ha la sua personale scala di valutazione: per esempio, un appassionato di fumetti Marvel tenderà a valutare con punteggi più alti i film Marvel, mentre un utente medio, che pur apprezza i film Marvel ma non è così tanto appassionato, li valuterà con punteggi nella media. Con i punteggi "puri", i due vettori utente sarebbero abbastanza diversi, per la distanza del coseno, ma se sottraiamo il rating medio dell'utente dai suoi singoli rating, otteniamo valori simili per questi film, e la similarità del coseno tra i due vettori è più vicina alla similarità reale tra i due utenti. La stessa logica è applicabile ai film: per questo motivo, scelgo di calcolare una nuova matrice di similarità, usando i rating normalizzati. In altre parole, dovrei ottenere calcoli top-k più accurati (i top-k film dovrebbero veramente simili al film dato). A causa delle similarità negative che potrebbero portare i rating predetti a essere fuori dal range $[1, 5]$, ho scelto di clipparli, cioè i rating sotto 1 saranno portati a 1, e quelli sopra 5 saranno tagliati a 5.

Risultati.

Table 3: RMSE del collaborative filtering item-item con diversi valori di k e rating normalizzati.

k	RMSE
2	1.0913592027780321
5	1.0100220088576446
10	1.004652443852321
20	1.0397412574115643
50	1.1353106405730728
100	1.2227231546441086

In realtà, i punteggi migliorano di circa 0.03 RMSE con i rating normalizzati (src_naive_cf/naive_norm_cf.py), per $k = 2, 5, 10$. Negli altri casi, stranamente, le prestazioni diminuiscono.

2.3.3 Collaborative filtering con bias

Per catturare completamente e considerare il bias soggettivo tra singoli utenti e singoli film, è meglio includere una forma di *bias* nei rating calcolati dal collaborative filtering. Possiamo riconoscere 2 tipi di bias:

- Bias dell'utente: quanto è "severo" l'utente con i rating? Sottraiamo semplicemente il rating medio globale dal rating medio dell'utente: se questo valore è positivo, l'utente è più generoso della media; se è negativo, l'utente è più severo della media.
- Bias del film: quanto è apprezzato il film dagli utenti? Come con il bias dell'utente, possiamo sottrarre il rating medio globale dal rating medio del film. Un valore positivo indica che il film è più apprezzato (e, teoricamente, è un "bel" film), e un valore negativo indica che il film è un "brutto" film.

Per calcolare il bias di un utente i verso il film j, è sufficiente aggiungere il bias dell'utente i e il bias del film j alla media globale e ottenere un punteggio **baseline**.

$$b_{ij} = \mu + b_i + b_j$$

Il rating finale calcolato dal collaborative filtering integra questo punteggio baseline nel punteggio del collaborative filtering di base.

$$r_{ij} = b_{ij} + \frac{\sum_k s_{jk}(r_{ik} - b_{ik})}{\sum_k s_{jk}}$$

Risultati.

Table 4: RMSE del collaborative filtering item-item con diversi valori di k, rating normalizzati e bias.

k	RMSE
2	1.0603730364833142
5	0.9845914585294856
10	0.9776019772042669
20	1.0049439563552225
50	1.0742920767167603
100	1.132446440492208

Con i bias (src_naive_cf/naive_biased_cf.py), i punteggi RMSE migliorano di circa 0.03 RMSE rispetto al CF senza bias, per $k = 2, 5, 10, 20$. Il miglior risultato è ancora $k = 10$, con 0.97 RMSE.

Dato che mi sarei aspettato risultati ben migliori, mi riserverò di ricalcolarli usando i bias e le similarità non normalizzate.

Risultati (similarità con rating non normalizzati).

Table 5: RMSE del collaborative filtering item-item con diversi valori di k, SENZA rating normalizzati e con bias.

k	RMSE
2	1.0578293613594116
5	0.9787971310190972
10	0.958036342585129
20	0.9542251364232185
50	0.960075653507239
100	0.965302178415238

Ora, con $k = 20$, il collaborative filtering con bias ha prestazioni quasi equiparabili a quelli di Cinematch, con RMSE pari a 0.9542 contro 0.9514. In realtà, questo algoritmo dovrebbe riuscire a superare Cinematch, attestandosi su circa 0.94 di RMSE; ma probabilmente quei punteggi consideravano il training set compreso di probe, e il quiz set come test set.

2.4 Weighted collaborative filtering

Almeno nel nostro caso, per superare la "barriera" di Cinematch (0.9514), bias e similarità precalcolate non sono stati sufficienti. Per questo motivo, la prima idea che può viene in mente è: e se non usassimo valori precalcolati, ma "pesi" appresi che minimizzano una funzione di perdita? Cioè, il tipico approccio di machine learning ai problemi di predizione. Lo scopo è mappare e catturare interazioni nascoste tra elementi che i nostri bias e similarità fissi non catturano.
Koren e Bell nel 2007 introdussero due modelli di collaborative filtering pesato:

- Un modello basato sul vicinato, come i precedenti, ma con pesi di interpolazione congiunti: per ogni elemento, i pesi catturano le relazioni tra esso e i top-K vicini.
- Un modello globale: i pesi catturano le relazioni complessive tra elementi; questo modello supporta anche l'uso di un vicinato limitato, ma la versione con le migliori prestazioni è la versione con vicinato "completo" (cioè k = numero di elementi).

Il secondo modello è quello con prestazioni migliori; mostreremo questo, nella versione completa.

2.4.1 GlobalNgbr

In questo modello, similarità e bias sono appresi con SGD. C'è anche un altro insieme di pesi, chiamati **pesi impliciti**. Questi pesi mappano, come dice il nome, relazioni "implicite" tra item, senza considerare i rating. In altre parole, il peso w_{AB} misura quanto il rating di A è statisticamente importante per calcolare il rating dall'utente all'item B, dato che l'utente ha valutato entrambi gli item. Un valore alto di w_{AB} indica che la presenza di A nei rating dell'utente aumenta il rating dell'utente verso B, e viceversa. L'implementazione originale di Koren e Bell usa SGD con SSE come funzione di loss. Dato un utente u e un item i, il rating è calcolato come segue:

$$\hat{r_{ui}} = b_{ui} + \sum_{j \in R(u)} w_{ij}(r_{ij} - b_{uj}) + c_{ij}$$

Risultati. Questo modello (global_ngbr/test_global_ngbr.py) è il primo a battere Cinematch di circa 0.03 RMSE: otteniamo un punteggio di 0.9228 RMSE, con 20 epoche, learning rate di 0.01 e regolarizzazione λ di 0.002. In "Advances in Collaborative Filtering", 5.1.2 e 5.1.3 , Koren e Bell affermano di aver ottenuto un

risultato di 0.9002 RMSE sul test set, usando l'intero training set. In realtà il learning rate usato da loro era 0.005, che io ho deciso di aumentare per motivi di tempo. La regolarizzazione è invece uguale a quella utilizzata da Koren e Bell.

2.5 Matrix factorization e fattori latenti

N.B. Tutte le implementazioni di modelli MF si trovano nella cartella "svd".

L'idea principale dietro questo tipo di modelli è rappresentare utenti e item con degli embeddings. Dato un parametro K, ogni utente/item è rappresentato con un vettore di K componenti. Ogni componente è chiamato *fattore latente*, dedotto dai dati di training. Questi modelli a "fattori latenti" associano delle caratteristiche latenti ad utenti/item e per questo i loro risultati non sono immediatamente spiegabili. Da un certo punto di vista, in questo scenario, potremmo vederli come generi/tipi di film, o cose simili.

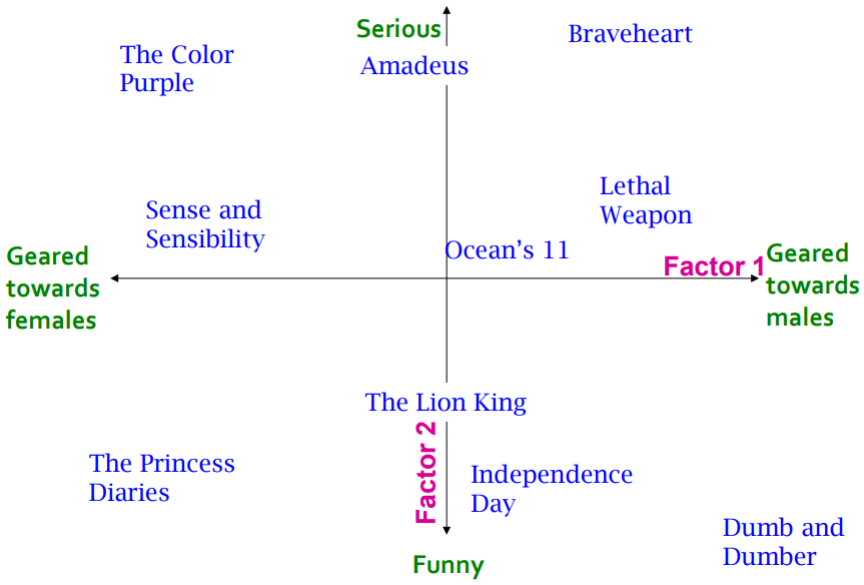

Figure 2: Un esempio giocattolo di "fattori latenti".

Tutti gli embedding di utenti/item impilati insieme formano una coppia di matrici chiamate $P \in \mathbb{R}^{M \times K}$, che rappresenta gli utenti, e $Q \in \mathbb{R}^{N \times K}$ che rappresenta gli item, dove M è il numero di utenti nel dataset e N è il numero di item. La versione di base di questo modello è chiamata "Funk SVD", dal creatore Simon Funk, e perché possiamo ricostruire l'intera matrice di utilità da queste due matrici, in maniera simile a come farebbe la SVD, anche se non si tratta veramente di una SVD ma bensì di un metodo supervisionato. Il rating dell'utente u all'item i è calcolato con il prodotto scalare tra l'embedding dell'utente P_u e l'embedding dell'item Q_i, e, ovviamente, gli embedding sono appresi attraverso la discesa del gradiente.

$$\hat{r_{ui}} = q_i^T p_u$$

Risultati. Stranamente, i risultati del Funk SVD di base sono molto peggiori dei risultati del GlobalNgbr, con la mia implementazione PyTorch che ottiene 0.993398 RMSE dopo 3 epoche ($K = 100$), e inizia a fare overfitting dopo la 4a epoca. Learning rate 0.001, ottimizzatore AdamW con regolarizzazione 0.01.

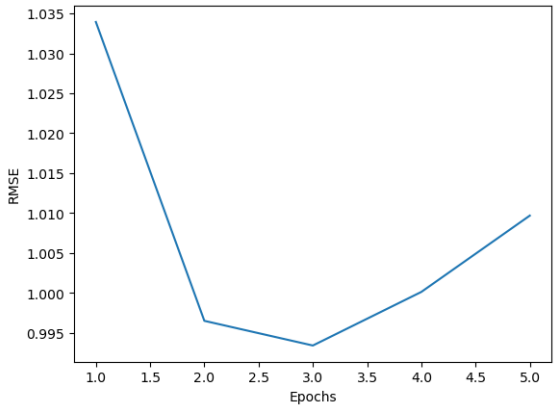

Figure 3: RMSE sul probe set con il modello Funk SVD.

2.5.1 SVD con bias

Koren e Bell hanno migliorato il Funk SVD originale aggiungendo gli stessi predittori baseline (addestrabili) già utilizzati nel modello GlobalNgbr.

$$\hat{r_{ui}} = \mu + b_i + b_j + q_i^T p_u$$

Risultati. L'aggiunta dei predittori baseline ha aiutato molto le prestazioni del modello. Il mio SVD con bias ha ottenuto 0.921859 RMSE con $K = 100$, dopo 2 epoche di addestramento. Come nel caso precedente, il modello inizia a fare overfitting presto, dopo la 3a epoca. Learning rate 0.001, ottimizzatore AdamW con regolarizzazione 0.01.

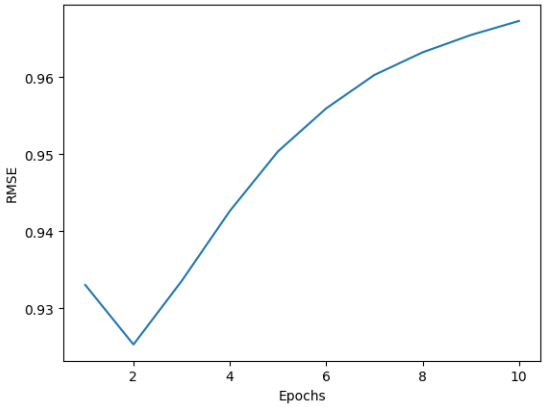

Figure 4: RMSE sul probe set con il modello SVD con bias.

In generale, con PyTorch e l'ottimizzatore AdamW (che non esisteva al tempo del Netflix Prize), tutti i modelli di tipo SVD convergono dopo 2 o 3 epoche, circa 1/2 ore di addestramento.

2.5.2 SVD++

Poco dopo l'introduzione di SVD, Koren, nel paper "Factorization Meets the Neighborhood", ha introdotto il modello SVD++, un miglioramento dell'SVD con bias che considera anche le relazioni implicite tra item, come il GlobalNgbr. Viene introdotta una nuova matrice $Y \in \mathbb{R}^{N \times K}$: per ogni r_{ui} viene calcolato anche un embedding implicito, sommando tutti gli embedding y_j, con $j \in R(u)$; in altre parole, questo embedding rappresenta la relazione implicita tra tutti e soli gli item valutati da u. Questo embedding somma è normalizzato da $|R(u)|^{-1/2}$ ($1/2$ è stato scelto da Koren con cross-validation), e sommato con p_u.

$$\hat{r_{ui}} = \mu + b_i + b_j + q_i^T(p_u + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j)$$

Risultati. L'SVD++ con $K = 50$ è quello che ha ottenuto i migliori risultati, con 0.915024 RMSE, offrendo un miglioramento di circa 0.006 rispetto ad SVD classico con $K = 100$. Learning rate 0.001, ottimizzatore AdamW con regolarizzazione 0.01.

2.6 Matrix factorization con dinamiche temporali

Analizzando la distribuzione dei rating e i timestamp, i vari team partecipanti alla challenge hanno notato come i rating medi, sia degli utenti sia dei film, cambiassero nel tempo. Questo era dovuto principalmente a:

- La popolarità di un item può cambiare (sia aumentare che diminuire), solitamente va a diminuire perchè l'effetto "hype" che si presenta al rilascio viene meno col tempo, ma potrebbe diminuire/aumentare anche a causa di vari avvenimenti esterni (ad esempio il salto di carriera di un attore, o la sua morte, o un avvenimento geopolitico).
- Un utente, da essere umano qual è, non rimane monolitico nei suoi gusti: col passare del tempo potrebbe iniziare ad apprezzare generi di film diversi da quelli che apprezzava fino a quel momento, oppure essere influenzato da sessioni di binge watching.

Per questo motivo, come descritto da Koren e Bell in "Advances in Collaborative Filtering" (3.3), hanno deciso di modellare questi fenomeni attraverso dei nuovi parametri.

2.6.1 Bias temporale per item

Queste variazioni, per gli item, avvengono solitamente in lassi di tempo più lunghi rispetto a quelle per il singolo utente, perchè sono espressioni di una tendenza generale. Per questo, Koren e Bell hanno deciso di dividere l'intero arco temporale del dataset in 30 sottoperiodi detti "bin", ognuno di circa 10 settimane (i rating vanno da ottobre 1998 a dicembre 2005). Questi bias dovevano tracciare i cambiamenti degli item bias nel tempo, e di conseguenza, dato un rating rilasciato in una certa data (a cui corrisponde un certo bin), riuscire a predire meglio usando l'item bias associato a quel bin.

Dato un giorno t (il numero di giorni passati dal primo rating in ordine temporale), l'item bias per un rating dato al giorno t è:

$$b_i(t) = b_i + b_{i,bin(t)}$$

2.6.2 Bias temporale per utente

Come detto prima, il cambiamento dei gusti ed abitudini di un singolo utente è più repentino rispetto a quanto può avvenire per un item. Per questo, hanno previsto un diverso parametro di bias utente per ogni giorno compreso nel dataset. Oltre a questo, viene calcolato anche un parametro di "drift", considerando il "giorno medio di rating" dell'utente, che permette di modellare meglio i cambi di abitudine.

$$dev_u(t) = sign(t - t_u)|t - t_u|^\beta$$

con $\beta = 0.4$ tramite cross validation. All'utente viene anche associato un parametro α_u, che ha il compito di pesare il drift, ed è appreso tramite discesa del gradiente.

Il modello baseline più semplice tenente conto delle dinamiche temporali per gli utenti si ottiene quindi:

$$b_u(t) = b_u + \alpha_u dev_u(t)$$

Una versione più complessa e precisa aggiunge il bias giornaliero $b_{u,t}$ descritto all'inizio:

$$b_u(t) = b_u + \alpha_u dev_u(t) + b_{u,t}$$

Il rating di baseline viene quindi calcolato così:

$$b_{ui}(t) = \mu + b_i + b_{i,bin(t)} + b_u + \alpha_u dev_u(t) + b_{u,t}$$

che ottiene un rispettabile punteggio (considerando che è solo baseline) di 0.9603 RMSE. Delle versioni successive ancora più complesse aggiungono un parametro di scaling (uno per ogni coppia utente, giorno) all'item bias che abbassa lo score a 0.9555 RMSE (quasi quanto Cinematch). In "The BellKor Solution to the Netflix Grand Prize" pubblicato dopo la fine della challenge Koren introduce anche un parametro di "frequenza", che misura il numero di rating dati dall'utente in un singolo giorno. E' proprio questo il parametro che dovrebbe misurare gli effetti di sessioni di "binge watching", che riesce ad avere un grandissimo effetto sulle performance, facendo ottenere a quello che rimane ancora un modello di baseline uno score di 0.9278 RMSE.

2.6.3 Embedding temporali per utente

Il SVD prevede due matrici di parametri: Q per gli embedding degli item, e P per gli embedding degli user. Ma dato che il comportamento dell'utente varia in funzione del tempo, bisogna avere un embedding diverso per ogni giorno, anche per lo stesso utente. Si avrà quindi una matrice 3D $P_{temp} \in \mathbb{R}^{M \times D \times K}$ dove M sono gli utenti, D i giorni e K i fattori.

$$p_u(t) = p_u + \alpha_u dev_u(t) + p_{u,t}$$

Arricchendo il modello SVD++ con questi embedding e con i bias temporali, si ottiene il modello **TimeSVD++**.

$$\hat{r_{ui}} = b_{ui}(t) + q_i^T(p_u(t) + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j)$$

Un modello molto più pesante a livello di parametri rispetto ai precedenti, e quindi difficile da implementare e far girare su una macchina di fascia medio bassa anche nel 2025, ma che riesce a migliorare i risultati di SVD++ di circa 0.01 RMSE, a parità di fattori latenti.

Risultati. Per le stesse ragioni sopracitate, non sono stato in grado di implementare la versione completa del TimeSVD++, per cui ho optato per l'implementazione di una versione più basica che includesse soltanto i bias temporali degli item, e i bias temporali degli utenti divisi per bin e NON per singoli giorni, disponibile nel file "svd/models.py" e chiamato "MiniTimeSVDpp". Il modello in questione con $K = 50$ ottiene sul probe set un RMSE di 0.913698, un miglioramento di circa 0.0013 rispetto ad SVD++ con un numero analogo di fattori latenti.

2.7 Modelli successivi e deep learning

2.7.1 Restricted Boltzmann Machines

Oltre ai modelli matrix factorization, che sono stati la base fondante del Netflix Prize, un'altra classe di modelli è stata presa in considerazione dai team partecipanti: le Restricted Boltzmann Machines, adattate appositamente per il task di collaborative filtering. Per mancanza di tempo, non sono riuscito ad implementarle, per cui non saranno incluse nel modello finale. L'implementazione di BellKor nel paper "The BellKor Solution to the Netflix Grand Prize" delle RBM riusciva ad ottenere risultati paragonabili ad un buon SVD++, circa 0.895 di RMSE.

2.7.2 AutoRec

Un modello all'apparenza molto interessante per il task di collaborative filtering attraverso predizione di rating espliciti è quello contenuto in "AutoRec: Autoencoders Meet Collaborative Filtering" pubblicato nel 2015. Il modello AutoRec non è altro che un semplice autoencoder che cerca di ricostruire, per ogni utente/item (dipende dalla versione del modello, user-based o item-based), tutto il vettore di rating preso dalla matrice di utilità, ovviamente anche i valori che in origine sono a zero. Gli autori, almeno per la versione item-based, rivendicavano prestazioni di poco superiori ad SVD.

La mia semplice implementazione in PyTorch della versione item-based (nella cartella "autorec") non è riuscita a replicare le prestazioni promesse, assestandosi a poco più di 1.06 di RMSE dopo quasi 200 epoche, con learning rate pari a 0.001 e 250 neuroni nel layer nascosto; per cui questo modello è stato scartato. Aumentare il numero di neuroni nell'hidden layer avrebbe richiesto una GPU con maggiore VRAM di quella a disposizione.

2.7.3 Factorization Machines

Una generalizzazione dei modelli di tipo Matrix Factorization sono le Factorization Machines, proposte nel 2010 (quindi dopo il Prize) in "Factorization machines" da Rendle e che riescono a modellare anche le relazioni tra feature aggiuntive sia degli utenti che degli item (esempio l'età, il sesso per l'utente e il genere, durata, attori per il film); in assenza di queste, il modello si comporta come un classico SVD. Per motivi temporali non sono riuscito ad integrare il mio dataset con dati aggiuntivi presi dai vari database di film su Internet quali IMDB e TMDB, ma uno degli sviluppi futuri più interessanti di questo progetto potrebbe essere proprio arricchire il dataset originale ed addestrare un modello FM per la predizione di rating espliciti (sono adatti sia a compiti di classificazione che di regressione). Esiste anche una versione deep learning, detta DeepFM e proposta nel 2017 in "DeepFM: A Factorization-Machine based Neural Network for CTR Prediction", anche se è indicata soprattutto per la predizione del click-through rate.

2.7.4 Neural CF

In presenza di rating impliciti (visto-non visto), un modello con buone prestazioni nel ranking è il Neural Collaborative Filtering, proposto nel 2017 nell'omonimo paper. Questo modello sfrutta in parallelo sia una MLP che la matrix factorization. Inizialmente crea due versioni delle matrici P e Q, una da usare in "SVD style", mentre l'altra da passare in input alla MLP. I vettori risultanti da entrambi i rami (l'SVD qui non produce uno scalare in output, moltiplica soltanto element-wise i vettori p_u, q_i) vengono concatenati, e un layer finale produce uno score in output, che indica la probabilità che l'utente accederà a quell'item. L'idea della MLP sarebbe quella di catturare anche relazioni non lineari tra user ed item.

Ho creato una semplice implementazione di questo modello, adattandolo al task di predizione di rating espliciti sostituendo semplicemente la loss con una classica MSE, ma ho ottenuto prestazioni praticamente identiche a quelle di SVD, per cui ho scelto di scartarlo.

2.7.5 Deep learning

In generale, i modelli allo stato dell'arte per raccomandazioni, come ad esempio i vari modelli graph-based (GCN, GraphSage), i modelli Two-Tower, oppure i modelli basati su transformer, non risolvono più il task di predizione esplicita di rating, per cui spesso risulta difficile adattarli ad esso, col rischio di non ottenere alcun risultato; e soprattutto rendono al meglio solo in presenza di feature aggiuntive.

3 Modello ensemble

Nonostante le buone prestazioni ottenute da TimeSVD++, il punteggio del Grand Prize era circa 0.02 RMSE più basso di quello ottenuto da quest'ultimo. Per questo motivo, i team BellKor, Pragmatic e BigChaos decisero di unire le forze e anche i loro modelli, per creare un ensemble che teoricamente avrebbe dovuto migliorare il punteggio finale rispetto ai modelli presi singolarmente, e quindi avvicinarli al milione di dollari. La strategia classica per creare una miscela di modelli diversi è pesare e sommare ogni singolo rating: per esempio, dati due modelli m_1, m_2, l'output combinato è

$$\lambda_1 m_1 + \lambda_2 m_2$$

con $\lambda_1, \lambda_2 \in [0, 1]$. Ma questo tipo di modelli ibridi è pesante e difficile da ottimizzare; il team BellKor's Pragmatic Chaos decise di usare dei gradient boosted decision trees per questo scopo. Come usarono i GBDT? Hanno fatto predire tutti i rating del probe set a tutti i modelli facenti parte dell'ensemble, e hanno creato un meta-dataset con queste predizioni come feature, e i rating reali come target, per addestrare il GBDT. L'intuizione dietro questo modello di blending è che l'ensemble di alberi dovrebbe imparare automaticamente quale modello predice meglio un certo gruppo di rating, e, quindi, sempre in linea teorica, dovrebbe riuscire a scegliere sempre il modello migliore per il caso specifico.

L'ensemble dietro al BellKor's Pragmatic Chaos conteneva centinaia di modelli, che comprendevano varianti di SVD, SVD++, TimeSVD++, GlobalNgbr, modelli baseline, collaborative filtering con vicinato, e Restricted Boltzmann Machines.

3.0.1 Mini-ensemble finale

Per motivi di tempo, e anche i motivi citati in precedenza, il mio ensemble finale sarà una versione molto "striminzita", composta da soli 8 modelli. Con un maggiore quantitativo di tempo a disposizione, sarebbe risultato molto interessante implementare anche modelli come RBM. Dato il solito problema di non poter testare i modelli sul quiz set, ho dovuto scegliere di dividere il probe in due parti uguali: una sarà utilizzata per addestrare il GBDT, e l'altra sarà utilizzata per il testing. I modelli inclusi nel mio mini-ensemble sono i seguenti:

- SVD con 50 fattori latenti, che sulla parte di probe set scelta per il testing del GBDT ottiene singolarmente 0.921673 di RMSE;
- SVD con 100 fattori latenti, che ottiene singolarmente 0.920966 di RMSE;
- SVD con 150 fattori latenti, che ottiene singolarmente 0.929917 di RMSE;
- SVD++ con 50 fattori latenti, che ottiene singolarmente 0.914338 di RMSE;
- SVD++ con 100 fattori latenti, che ottiene singolarmente 0.919281 di RMSE;
- SVD++ con 150 fattori latenti, che ottiene singolarmente 0.924341 di RMSE;
- GlobalNgbr come è stato presentato in precedenza, che ottiene singolarmente 0.924254 di RMSE.
- MiniTimeSVDpp con 50 fattori latenti, che ottiene singolarmente 0.912838 di RMSE.

Risultati. Tra quelli testati sul campo (ensemble/ensemble.py), il GBDT che ottiene il miglior risultato, pari a 0.892 di RMSE, contiene 25 estimatori, con una profondità massima di 5, un learning rate di 0.2 ed un subsampling per l'addestramento dei singoli estimatori pari ad 1 (in altre parole, ognuno dei 25 estimatori viene addestrato con l'intero training set). Non è neanche lontanamente paragonabile a quanto dovrebbe ottenere un modello che pretende di competere per il Netflix Prize, dato il tempo limitato, ma è comunque un buon esempio didattico del funzionamento dei migliori modelli partecipanti alla challenge, e soprattutto fa capire che l'idea dell'ensemble migliora sensibilmente le prestazioni rispetto ad ognuno dei modelli singoli.

4 Conclusioni

La sfida è stata (ovviamente) non superata, anche a causa di tempo e risorse limitati a mia disposizione, ma questo progetto mi ha permesso di poter approfondire oltre al livello offerto dal corso le basi fondanti dei sistemi di raccomandazione, come il paradigma Collaborative Filtering, che cerca di raccomandare un item ad un utente basandosi sui dati disponibili degli altri utenti ed item; ed esplorare il funzionamento dei principali modelli CF, come i modelli basati sul vicinato e i modelli Matrix Factorization. Il requisito di doverli implementare manualmente mi ha permesso di capirli più in profondità. Durante lo studio della soluzione BellKor per il Grand Prize, sono venuto a conoscenza del loro metodo per blending di più modelli e la creazione di un modello ibrido, un metodo più intuitivo rispetto a quello presentato nel corso, e capace di migliorare in maniera considerevole le prestazioni.

Una delle parti più interessanti dell'intero progetto è stata l'esplorazione, anche se in maniera molto superficiale, dello stato dell'arte nel campo della raccomandazione, e dei modelli usati in campo industriale dalle grandi aziende tech. I modelli SOTA potrebbero proprio essere parte integrante degli sviluppi futuri per questo progetto, quali l'integrazione di feature aggiuntive nel dataset, l'implementazione di factorization machines e la loro inclusione dentro l'ensemble.