



UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

Alessandro Sebastiano Catinello

Personal Content Based
Recommendation System for Netflix

RELAZIONE PROGETTO FINALE

Relatore: Prof. Antonino Furnari

Anno Accademico 2022 - 2023

Abstract

Lo scopo di questo progetto è la realizzazione di un sistema di raccomandazione dei prodotti disponibili sulla piattaforma di streaming Netflix basato sul contenuto e sulle preferenze degli utenti. Il sistema è stato realizzato allo scopo di offrire un sistema di raccomandazione personale, alternativo al sistema proprietario offerto da Netflix. L'obiettivo è quindi quello di ottenere un sistema indipendente da bias pubblicitari.

Si è approcciato inoltre il problema della raccomandazione come un problema di regressione, considerando anche scenari in cui non si hanno molte valutazioni degli utenti.

Sono stati utilizzati due dataset differenti: il primo è stato creato tramite scraping per ottenere i metadata dei film e delle serie TV, successivamente sfruttati per la rappresentazione informativa dei dati; il secondo è il Netflix Prize Dataset, contenente le valutazioni degli utenti.

Il modello su cui si è concentrato il progetto è il K-NN Regressor che, in base a dove vengono mappati i film nello spazio, predice il rating dell'utente a un film non visto. I risultati ottenuti non sono ottimali ma comunque accettabili, considerando anche il modello è stato allenato su un dataset con pochi dati.

Si nota inoltre che la rappresentazione dei film utilizzata non è ottimale, se essa fosse adeguata porterebbe sicuramente risultati migliori.

Indice

1	Introduzione	4
2	Strumenti Utilizzati	6
2.1	Sistemi di raccomandazione	6
2.1.1	Sistemi Content Based	10
	Profilo di un oggetto	10
	Profilo di un utente	11
	Similarità	12
2.1.2	Collaborative Filtering	14
	Similarità	15
	Predizioni	16
2.2	Regressione	18
2.2.1	KNN-Regressor	19
	1-Nearest Neighbors	20
	k-Nearest Neighbors	20
2.2.2	Regressione Lineare	22
	Regressione Lineare Semplice	22
	Regressione Lineare Multipla	24
	Apprendimento e Discesa del Gradiente	24
2.2.3	Regressione Ordinale	28
2.3	Misure di Performance	30
2.3.1	Root Mean Squared Error	30
2.3.2	Mean Absolute Error	31
2.4	Word Embedding	32

<i>INDICE</i>	3
2.5 Riduzione Dimensionale	34
2.5.1 PCA - Principal Component Analysis	34
2.5.2 Standard Scaler	36
3 Dati utilizzati	37
3.1 Dataset	37
3.1.1 Catalogo Netflix	38
3.1.2 Netflix Prize Dataset	39
3.2 Pre-processing	40
3.2.1 Catalogo Netflix	40
Embeddings delle trame	40
Embeddings Metadata e Dimensionalità	41
3.2.2 Netflix Prize Dataset	42
4 Approccio	43
4.1 Approccio Qualitativo	43
4.2 Approccio Quantitativo	44
4.2.1 Regressione	45
4.2.2 Misure di Performance	47
Correzione tramite Bias	47
5 Esperimenti e Risultati	50
5.1 Risultati Qualitativi	50
5.1.1 Test qualitativi tra film	50
5.1.2 Test qualitativi con profilo utente	51
5.2 Risultati Quantitativi	52
5.2.1 K-NN Regressor	53
Parametri di configurazione	53
Scelta del parametro K	54
5.2.2 Confronto tra modelli	55
Conclusione	58
Bibliografia	60

Capitolo 1

Introduzione

Nei tempi odierni, i sistemi di raccomandazione rivestono un ruolo essenziale nel settore del marketing. La loro importanza è dovuta al fatto che, grazie a questi sistemi, è possibile predire le preferenze degli utenti, dando quindi la possibilità di proporre loro prodotti che potrebbero suscitare loro interesse. Questi algoritmi hanno quindi trasformato il modo con cui le aziende interagiscono con i propri clienti: esse possono infatti indirizzare offerte mirate, migliorando l'esperienza d'acquisto e incoraggiando la fedeltà del cliente.

I sistemi di raccomandazione trovano un'ottima applicazione negli store digitali dove, a differenza dei negozi fisici, non c'è una limitazione di spazio per esporre i prodotti e non si è quindi limitati a proporre solo i prodotti più venduti. Essi permettono quindi di proporre anche articoli poco noti ma coerenti con il profilo dell'utente.

Il medesimo ragionamento è valido anche per i servizi di streaming, come Netflix, che offrono un vasto catalogo di contenuti. Ogni utente ha preferenze differenti e la piattaforma stessa opera già con un suo sistema di raccomandazione. [1]

Tale sistema di raccomandazione è però basato su dati la cui provenienza è sconosciuta e la raccomandazione offerta è spesso condizionata da una forte componente pubblicitaria, che spinge l'utente a guardare contenuti che sono stati prodotti da Netflix stessa o che sono stati aggiunti alla piattaforma di recente.

Questo progetto si pone quindi l'obiettivo di cogliere le preferenze dell'utente, creando un sistema di raccomandazione personale basato sul contenuto in grado di consigliare i film e serie TV, presenti nel catalogo Netflix, più adatti al suo profilo. Il catalogo è stato ottenuto grazie a uno script di web Scraping scritto in Python e la qualità delle raccomandazioni è stata misurata basandosi su ratings reali di utenti di Netflix.

Il resto della tesi è sviluppato come segue:

- Nel capitolo 2 si introduce il problema della raccomandazione e si descrive la teoria delle tecnologie utilizzate.
- Nel capitolo 3 si descrivono i dati utilizzati per il progetto, fornendo una descrizione dettagliata dei dataset e delle operazioni di pre-processing effettuate.
- Nel capitolo 4 si descrivono i modelli utilizzati per la raccomandazione, sia qualitativi che quantitativi, e si forniscono i dettagli implementativi.
- Nel capitolo 5 si descrivono i risultati ottenuti, sia in termini di performance che di qualità delle raccomandazioni.
- Nel capitolo 6 si descrivono le conclusioni del progetto e si forniscono spunti per possibili sviluppi futuri.

Capitolo 2

Strumenti Utilizzati

Si elenca di seguito la teoria necessaria per comprendere il funzionamento dei sistemi di raccomandazione e le tecnologie utilizzate per la realizzazione del progetto.

2.1 Sistemi di raccomandazione

I sistemi di raccomandazione sono un'ampia classe di applicazioni Web che si pongono l'obiettivo di predire le preferenze di un utente per oggetti specifici, sulla base delle preferenze espresse in passato da esso o da altri utenti simili. L'idea alla base è così definita:

1. Un utente interagisce con una serie di oggetti fisici o virtuali
2. Il sistema di raccomandazione analizza le preferenze dell'utente e ne crea un profilo
3. Tale profilo è utilizzato per la creazione di un modello che predirà le preferenze dell'utente e gli suggerirà oggetti che potrebbero interessargli
4. Le raccomandazioni vengono inoltre utilizzate per future predizioni in modo tale da migliorare il modello e consigliare oggetti che l'utente stesso potrebbe ancora non conoscere, ma al quale potrebbe essere interessato

Esistono diversi tipi di sistemi di raccomandazione, i più comuni sono:

- Collaborative Filter: si basano sulle preferenze di un gruppo di utenti simili a quello considerato. Se due utenti hanno espresso preferenze simili in passato, è probabile che anche in futuro esprimano preferenze simili. Il sistema di raccomandazione quindi suggerirà all'utente oggetti che sono stati apprezzati da altri utenti simili a lui.
- Sistemi Content Based: si basano sulle caratteristiche degli oggetti. Se un utente ha espresso preferenze per oggetti con determinate caratteristiche, il sistema di raccomandazione suggerirà oggetti con caratteristiche simili.

Problema della Long Tail

Il successo dei sistemi di raccomandazione è principalmente dovuto al fatto che offrono un'ottima soluzione al problema della Long Tail, ovvero la difficoltà di vendere prodotti poco noti. Tale fenomeno infatti è osservabile nei negozi fisici, dove la quantità di prodotti venduti è limitata dallo spazio fisico disponibile e dove vengono esposti solo i prodotti più popolari e di più facile vendita. Tale approccio però non copre tutto il mercato, in quanto non c'è alcun controllo sulle preferenze degli utenti, che potrebbero essere interessati a prodotti non esposti. [2]

D'altra parte, gli store digitali non hanno limitazioni di spazio e possono quindi soddisfare le richieste di tutti gli utenti, purché si conoscano le loro preferenze.

Il fenomeno è descritto in figura 2.1. Sull'asse verticale è rappresentato il numero di vendite o popolarità, mentre sull'asse orizzontale sono ordinati i prodotti per popolarità. I negozi fisici forniscono solo i prodotti più popolari sulla sinistra del grafico, mentre i negozi digitali possono fornire anche i prodotti meno popolari rappresentati dalla Long Tail nel grafico.

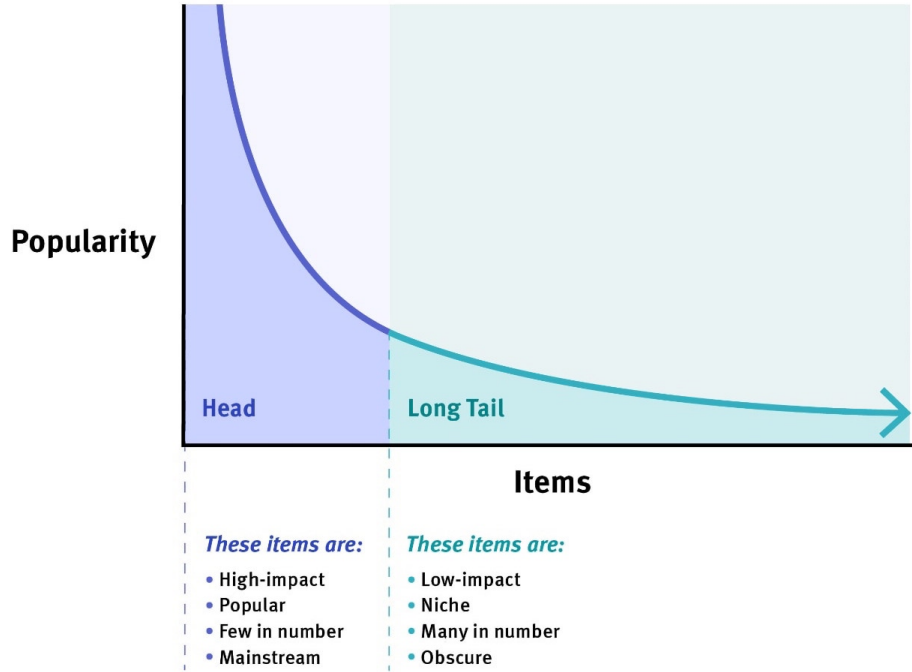


Figura 2.1: Il fenomeno della Long Tail.

Definizione del problema

Un sistema di raccomandazione ha il compito di predire l'interesse di un utente per un oggetto.

Definito quindi l'insieme X degli utenti e l'insieme S degli oggetti, l'obiettivo è quello di assegnare un numero reale, rappresentante il voto o il grado di interesse, per ogni coppia di elementi $(x, s) \in X \times S$.

Si definisce inoltre l'insieme R dei voti come un insieme ordinabile, ovvero un insieme in cui è possibile definire un ordinamento tra i suoi elementi. Gli elementi di R possono essere di diverso tipo:

- Numeri reali normalizzati: $R = [0, 1] \in \mathbb{R}$
- Valori booleani: $R = \{0, 1\}$ o $R = \{true, false\}$
- Valori ordinali: $R = \{1, 2, 3, 4, 5\}$

Da tali insiemi si può definire ora la matrice M detta “di utilità” dove ogni riga corrisponde a un generico utente x_i , ogni colonna rappresenta un oggetto

s_j e ogni cella m_{ij} , che corrisponde a una coppia $(x, s) \in X \times S$, contiene il voto $r(x_i, s_j) \in R$ assegnato dall'utente x_i all'oggetto s_j (come in figura 2.2). L'obiettivo è quindi quello di trovare una funzione $u : M \rightarrow R$ che assegni un voto ad ogni cella $m \in M$ in modo tale che $u(x, s) \approx r(x, s)$ per ogni $m \in M$, potendo così predire il voto che un utente darebbe ad un oggetto che non ha ancora valutato e popolando quindi la matrice M .

Utility Matrix: $U = [r(x_i, s_j)]_{i,j}$ (sparse), $r(x_i, s_j) \in R$

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Figura 2.2: Esempio di matrice di utilità

È previsto che la matrice di utilità sia sparsa, ma in un primo momento sono necessari dei dati per poterla popolare anche se in modo parziale. Acquisire questi dati può essere difficile in quanto gli utenti potrebbero non essere disposti a fornire informazioni su di loro o sulle loro preferenze. Questo problema è noto come cold start problem.

Esistono diversi modi per risolvere il problema:

- Richiedere agli utenti esplicitamente di fornire informazioni su di loro o sulle loro preferenze
- Apprendere le preferenze degli utenti in modo implicito, osservando il loro comportamento

L'approccio esplicito è spesso utilizzato ma poco efficace: le informazioni che si ottengono possono essere biased, in quanto provengono solo dagli utenti che vogliono fornirle o addirittura solo da utenti che lasciano recensioni positive.

L'approccio implicito è invece più semplice da applicare ma meno informativo: spesso si utilizza solo per estrarre informazioni riguardanti l'interesse, di

natura quindi booleana.

2.1.1 Sistemi Content Based

I sistemi di raccomandazione Content Based si basano sull'idea secondo la quale, se un utente ha espresso interesse per un oggetto, allora probabilmente sarà interessato anche ad oggetti simili. Se, ad esempio, un utente gradisce un film di un regista in particolare allora con alta probabilità gradirà anche altri film dello stesso regista.

La struttura alla base di un sistema di raccomandazione Content Based è composta da tre parti:

- Si creano i profili degli oggetti, ovvero si estraggono le caratteristiche degli oggetti
- Dalla matrice di utilità si estraggono le preferenze degli utenti e si crea il profilo di ogni utente
- Si confrontano i profili degli utenti con i profili degli oggetti per trovare le raccomandazioni in base alla similarità

Profilo di un oggetto

I sistemi content based necessitano di un modo per rappresentare gli oggetti. Per fare ciò si creano i profili degli oggetti, ovvero si estraggono le caratteristiche identificative in base alla tipologia di oggetto che si sta considerando. Nei casi più semplici quindi, il profilo di un oggetto consiste semplicemente nelle caratteristiche che lo identificano, facilmente recuperabili. Ad esempio, nel caso di un film, il profilo potrebbe essere composto da:

- Titolo
- Regista
- Genere

- Attori
- Anno di uscita

In altri casi, come testo o immagini, invece l'estrazione delle caratteristiche principali non è diretta come per film, libri o musica. In questi casi si utilizzano tecniche di feature extraction per estrarre le caratteristiche principali.

Ad esempio, nel caso di un testo, spesso si utilizza la tecnica del TF-IDF (Term Frequency - Inverse Document Frequency) che consiste nel calcolare la frequenza di ogni parola all'interno del testo e moltiplicarla per un peso che dipende dall'inverso della frequenza della parola all'interno del corpus. Applicato ciò, basterà selezionare le n parole con peso maggiore per ottenere le caratteristiche principali del testo.

Invece, nel caso di immagini, ci si basa spesso sui "tags" dell'immagine in questione, ovvero parole che ne descrivono il contenuto. Questi tags possono essere inseriti manualmente dall'utente o estratti automaticamente da un algoritmo di machine learning di feature extraction. La qualità dei tags è quindi molto variabile e dipende da chi li inserisce o da come sono stati estratti.

Una volta ottenute le caratteristiche principali di un oggetto, si può creare il suo profilo come un vettore di caratteristiche dove la presenza della caratteristica i -esima è rappresentata da un valore binario.

Un altro approccio valido è quello di utilizzare tecniche più complesse per la rappresentazione dei dati, come BOW (Bag Of Words) per i testi o BOVW (Bag Of Visual Words) per le immagini, al fine di migliorare la qualità della raccomandazione.

Profilo di un utente

Una volta creati i profili degli oggetti, si devono creare i profili degli utenti. Questi profili sono necessari per riassumere le preferenze degli utenti in modo da poterli poi confrontare con i profili degli oggetti. L'obiettivo è quindi

quello di creare un vettore di caratteristiche, della stessa dimensione di quello degli oggetti, che quantifichi le preferenze dell'utente.

Per fare ciò, si utilizza la matrice di utilità M che contiene i voti assegnati dagli utenti agli oggetti. La generazione del profilo di un utente varia in base alla tipologia di voto presente nella matrice di utilità.

Nel caso di voti binari, il metodo più semplice è quello di calcolare una media dei voti presenti nella matrice in corrispondenza della riga dell'utente in questione:

$$profile(x_i) = \frac{1}{\sum_j U(x_i, s_j)} \sum_{j=1}^n U(x_i, s_j) \cdot profile(s_j) \quad (2.1)$$

Invece, nel caso di voti reali, è possibile utilizzare un metodo più complesso che migliora la qualità del profilo. Un possibile approccio è quello di normalizzare il profilo con la media delle valutazioni: così facendo le valutazioni sotto la media otterranno un peso negativo e quelle sopra la media un peso positivo. In questo modo si ottiene un profilo che tiene conto anche della tendenza dell'utente a dare voti alti o bassi.

Si definisce quindi la media delle valutazioni:

$$\mu_i = \frac{1}{\sum_j [U(x_i, s_j) \neq 0]} \sum_{j=1}^n U(x_i, s_j) \quad (2.2)$$

E si calcola il profilo normalizzato:

$$profile(x_i) = \frac{1}{\sum_j [U(x_i, s_j) \neq 0]} \sum_{j=1}^n (U(x_i, s_j) - \mu_i) \cdot profile(s_j) \quad (2.3)$$

Similarità

Una volta calcolati i profili degli oggetti e degli utenti, si possono confrontare per trovare le similitudini. In particolare, si calcola la similarità tra il profilo dell'utente e il profilo di ogni oggetto. Se i due profili sono simili allora l'oggetto in questione sarà raccomandato all'utente.

Per calcolare la similarità tra due profili si utilizzano diverse metriche. Le più comuni sono:

- Similarità del Coseno: calcola la similarità tra due vettori come il coseno dell'angolo tra i due vettori. Tale valore è compreso tra -1 e 1 dove 1 indica che i due vettori sono identici, -1 indica che i due vettori sono opposti e 0 indica che i due vettori sono ortogonali.

La similarità del coseno tra due vettori x e y è definita come:

$$\text{cossim}(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|} = \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}} \quad (2.4)$$

- Distanza Euclidea: calcola la distanza tra due vettori come la radice quadrata della somma dei quadrati delle differenze tra i valori dei due vettori. Rappresenta semplicemente la distanza spaziale tra i due vettori: minore è la distanza, maggiore è la similarità.

$$\text{euclidean}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2.5)$$

$$\text{eucsim}(x, y) = \frac{1}{1 + \text{euclidean}(x, y)}$$

Pro e Contro

I sistemi di raccomandazione basati sul contenuto presentano diversi vantaggi, tra questi:

- Trasparenza: il sistema è in grado di spiegare le raccomandazioni fornite all'utente in base alle caratteristiche degli oggetti e degli utenti.
- Facilità di implementazione: il sistema è semplice da implementare e non richiede un grande sforzo computazionale.
- Indipendenza: il sistema non ha bisogno di informazioni riguardanti altri utenti o oggetti per fornire raccomandazioni.

- Problema del First Rater mancante: il sistema è in grado di fornire raccomandazioni anche per nuovi utenti o nuovi oggetti in quanto non dipende da altri utenti.

Tuttavia, presentano anche diversi svantaggi:

- Over-Specialization: il sistema tende a raccomandare oggetti simili a quelli già valutati dall'utente, non fornendo quindi raccomandazioni nuove e diverse. Gli utenti, inoltre, potrebbero avere più interessi e preferenze, ma il sistema non è in grado di tenerne conto in quanto non tiene traccia delle valutazioni qualitative di altri utenti che potrebbero migliorare la granularità delle raccomandazioni.
- Difficoltà di rappresentazione: non è sempre facile rappresentare gli oggetti nel modo ideale.
- Difficoltà nell'aggiungere nuove proprietà: se si vuole aggiungere una nuova proprietà ad un oggetto, allora è necessario ricalcolare tutti i profili degli oggetti e dell'utente. Tale operazione diventa ancora più complessa se l'utente non ha espresso preferenze specifiche rispetto alla nuova proprietà.

2.1.2 Collaborative Filtering

I sistemi di raccomandazione collaborative filter hanno invece un approccio totalmente diverso alla raccomandazione. Si basano sull'idea secondo la quale utenti simili avranno gusti simili, concentrandosi quindi sugli utenti e non sugli oggetti. Il sistema cerca di trovare utenti simili all'utente preso in campione e raccomanda gli oggetti valutati positivamente da questi utenti.

La struttura di un sistema di raccomandazione collaborative filter è la seguente:

1. Si seleziona un utente generico x_i al quale si vuole fornire una raccomandazione.

2. Si calcola la similarità tra x_i in questione e tutti gli altri utenti.
3. Si selezionano i k utenti più simili all'utente x_i .
4. Si popolano i voti mancanti dell'utente x_i con i voti degli utenti selezionati.

Similarità

Il calcolo della similarità tra i vari utenti è quindi fondamentale per il funzionamento del sistema. Anche in questo caso esistono diversi approcci al calcolo della similarità, ognuna in base al tipo di dato utilizzato per i voti nella matrice di utilità.

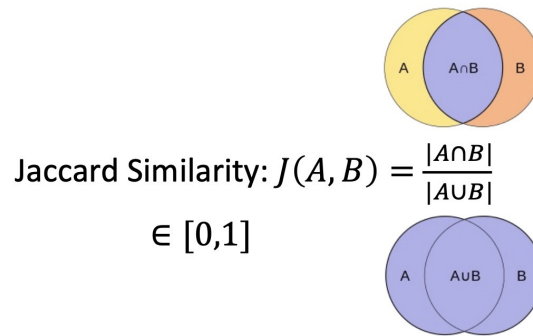


Figura 2.3: Similarità di Jaccard

In caso di dati binari, dove quindi il voto rappresenta solo se l'oggetto piace o no all'utente, si utilizza la similarità del Jaccard:

$$jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|} \quad (2.6)$$

La similarità di Jaccard (figura 2.3) è un numero compreso tra 0 e 1, dove 1 indica che le preferenze tra due utenti sono identiche, 0 indica invece che non hanno alcun elemento in comune. Tale metrica è applicabile solo ai dati binari in quanto è strettamente legata alla cardinalità degli insiemi. Usare la similarità di Jaccard con dati non binari porterebbe a grandi perdite di

informazioni, in quanto non tiene conto del valore dei voti e quindi del livello di gradimento dell'utente per gli oggetti votati.

In caso di dati non binari, dove quindi il voto rappresenta il livello di gradimento dell'utente per l'oggetto, si hanno molte più informazioni a disposizione per calcolare la similarità tra gli utenti. Si può quindi utilizzare la similarità del coseno, in questo caso più informativa rispetto alla similarità di Jaccard.

È necessario però normalizzare i voti in quanto il voto 0 potrebbe indicare che l'utente non ha ancora valutato l'oggetto oppure che l'utente non ha gradito l'oggetto. Per fare ciò, si sottrae ad ogni voto il valore medio dei voti dell'utente, così facendo i voti negativi saranno quelli che l'utente ha espresso ma che non ha gradito e i voti positivi saranno quelli che l'utente ha espresso e che ha gradito.

Predizioni

L'obiettivo finale del sistema di raccomandazione è quello di predire i voti mancanti per ogni utente. Per fare ciò i sistemi collaborative filter possono seguire due approcci:

- User - User: si basa sulla similarità tra gli utenti. Si selezionano gli utenti più simili all'utente in questione e si predicono i voti mancanti in base ai voti degli utenti selezionati.
- Item - Item: si basa sulla similarità tra gli oggetti. Si selezionano gli oggetti più simili all'oggetto preso in considerazione al quale l'utente non ha ancora espresso il voto e si predice il voto mancante in base ai profili degli oggetti selezionati.

User - User

Si considera un utente x_i e un item s_j per cui l'utente in questione non ha espresso alcun voto. Si vuole quindi predire il voto che l'utente darebbe

all'item s_j .

Per fare ciò, si crea l'insieme $N \subset K$, con K insieme degli utenti $\{x_k\}_{k=1}^{|K|}$, che hanno espresso un voto per l'oggetto s_j e sono più simili all'utente x_i . Per ottenere quindi la predizione basterà calcolare la media dei voti degli utenti in N :

$$u(x_i, s_j) = \frac{1}{|K|} \sum_{x_k \in N} r(x_k, s_j) \quad (2.7)$$

In aggiunta, si può anche introdurre la similarità tra gli utenti per pesare i voti degli utenti in N :

$$u(x_i, s_j) = \frac{\sum_{x_k \in N} \text{cossim}(\text{profile}(x_i), \text{profile}(x_k)) \cdot r(x_k, s_j)}{\sum_{x_k \in N} \text{cossim}(\text{profile}(x_i), \text{profile}(x_k))} \quad (2.8)$$

Item - Item

Un approccio differente alla predizione invece è quello Item - Item [3]. Si considera un utente x_i e un item s_j per cui l'utente in questione non ha espresso alcun voto.

La predizione comincia calcolando il profilo dell'item: si utilizza la colonna della matrice di utilità corrispondente all'item s_j (tutti i voti che l'item s_j ha ricevuto dagli utenti) e la si normalizza con la media dei voti. Si costruisce ora l'insieme N di oggetti più simili a s_j e votati dall'utente x_i . Si predice quindi il voto dell'utente x_i per l'item s_j come la media dei voti dell'utente x_i per gli oggetti in N :

$$u(x_i, s_j) = \frac{\sum_{s_k \in N} \text{cossim}(\text{profile}(s_j), \text{profile}(s_k)) \cdot r(x_i, s_k)}{\sum_{s_k \in N} \text{cossim}(\text{profile}(s_j), \text{profile}(s_k))} \quad (2.9)$$

All'atto pratico, l'approccio Item - Item è più efficace dell'approccio User - User. Questo perchè gli utenti tendono ad avere gusti particolari ed è più semplice trovare item simili tra loro piuttosto che utenti perfettamente simili tra loro. Inoltre, spesso sono presenti più item che utenti e si hanno quindi più esempi con il quale lavorare.

Pro e Contro

I sistemi di raccomandazione basati su collaborative filter hanno il vantaggio di lavorare con tutti gli oggetti presenti nel sistema, anche quelli nuovi. Inoltre, non è necessaria una fase di selezione ed estrazione delle caratteristiche degli oggetti.

Data però la loro natura, essi hanno bisogno di un grande numero di utenti e voti per poter funzionare correttamente. Inoltre, se un utente ha espresso pochi voti, la similarità tra gli utenti sarà bassa e la predizione sarà poco accurata. I sistemi collaborative filteri soffrono molto infatti del problema del cold start, ovvero non è possibile raccomandare oggetti a nuovi utenti di cui si hanno poche informazioni.

2.2 Regressione

Si definisce regressore una funzione:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

dove n è la dimensionalità del vettore di input e m è la dimensionalità dello spazio di output. Si ha quindi un vettore di input $x \in \mathbb{R}^n$, uno di output $y \in \mathbb{R}^m$ che rappresenta il vettore di verità di riferimento e infine un vettore $\hat{y} \in \mathbb{R}^m$ che rappresenta il vettore di output predetto dal regressore. In sintesi si ha: $\hat{y} = f(x)$.

Il vettore in input x è solitamente un vettore rappresentativo dei dati, risultante da una fase di feature extraction.

Si definisce il dataset $D = \{(x^{(j)}, y^{(j)})\}_j$ come l'insieme dei dati a disposizione. In D ogni coppia di elementi $(x^{(j)}, y^{(j)})$ è formata da un vettore di input $x^{(j)}$ e un vettore di verità $y^{(j)}$. Si definiscono ora i vari sottoinsiemi del dataset utili per l'addestramento e la validazione del regressore:

- Training set: $TR = \{(x^{(j)}, y^{(j)})\}_j$ utilizzato per addestrare il regressore

- Validation set: $VA = \{(x^{(j)}, y^{(j)})\}_j$ utilizzato per validare il regressore
- Test set: $TE = \{(x^{(j)}, y^{(j)})\}_j$ utilizzato per testare il regressore

Un regressore è quindi un algoritmo che, dato un training set TR , approssima la funzione f in modo tale che $f(x^{(j)}) \approx y^{(j)}$ per ogni $(x^{(j)}, y^{(j)}) \in TR$. Per fare ciò solitamente si ha un approccio di tipo supervisionato, ovvero si utilizza il training set TR per addestrare il regressore semplicemente eseguendo l'algoritmo sui vari vettori di input e cambiando i parametri interni in base alle predizioni ottenute, fino a quando non si minimizza la funzione di errore scelta. La funzione di errore è una funzione che misura la differenza tra il vettore di verità $y^{(j)}$ e il vettore di output predetto $\hat{y}^{(j)}$.

Esistono vari algoritmi di regressione che si differenziano per il modo in cui approssimano la funzione f .

2.2.1 KNN-Regressor

Dati gli elementi $x^{(j)} \in TR$, è possibile immaginare che ogni elemento $x^{(j)}$ sia un punto nello spazio \mathbb{R}^n . Se ne riporta un esempio binario nello spazio bidimensionale in figura 2.4.

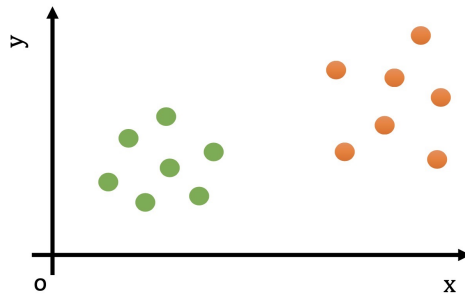


Figura 2.4: Esempio di punti nello spazio \mathbb{R}^2

Questa mappatura deve essere tale che punti vicini nello spazio \mathbb{R}^n siano anche simili tra loro.

1-Nearest Neighbors

Da tale premessa si può quindi definire l'approccio Nearest Neighbors (NN). L'algoritmo infatti si basa sulla similarità tra i punti nello spazio \mathbb{R}^n e fa parte degli algoritmi di apprendimento lazy: la fase di training non consiste nell'eseguire l'algoritmo e cambiare i parametri interni, ma consiste semplicemente nel memorizzare il training set TR .

Dato un punto $x^{(j)}$, in fase di predizione, basterà quindi trovare il punto $x^{(i)}$ più vicino a $x^{(j)}$ contenuto in TR , secondo una certa metrica di vicinanza o similarità, e usare il vettore di verità associato a $y^{(i)}$ come predizione in output per il punto $x^{(j)}$.

Definita quindi una funzione di similarità d come la distanza euclidea:

$$d(x, x') = \|x - x'\|_2 = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2} \quad (2.10)$$

O come la similarità del coseno:

$$d(x, x') = 1 - \frac{x \cdot y}{\|x\| \cdot \|y\|} = 1 - \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_i y_i^2}} \quad (2.11)$$

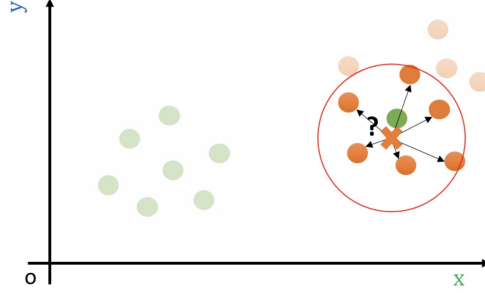
Si definisce la funzione di predizione che restituisce il vettore di verità dell'elemento più vicino a x' :

$$f(x') = \arg_y \min\{d(x, x') \mid (x, y) \in TR\} \quad (2.12)$$

Questo algoritmo è detto 1-NN in quanto considera solo il punto più vicino a x' . Ricade inoltre nella categoria di algoritmi non parametrici in quanto non ha parametri interni da addestrare, considerando anche la natura dell'addestramento lazy.

k-Nearest Neighbors

L'utilizzo però di un solo elemento come vicino per la predizione può portare a perdite di informazioni oltre che a predizioni errate nel caso in cui il punto

**Figura 2.5:** Esempio di k -NN

più vicino sia un outlier.

Per risolvere ciò, è possibile generalizzare l'algoritmo considerando i k punti più vicini a x' . In questo caso il parametro k diventa un iperparametro dell'algoritmo, ovvero un parametro scelto dall'utente a priori o ottimizzato tramite cross-validation.

Dato quindi un punto x' (come in figura 2.5), si definisce il suo intorno $N_k(x')$, centrato nel punto, come l'insieme dei k punti più vicini a esso:

$$N_k(x') = \{x \in TR \mid d(x, x') \leq \epsilon_k(x')\} \quad (2.13)$$

Dove con $\epsilon_k(x')$ si indica la distanza:

$$\epsilon_k(x') = \varepsilon \text{ tale che } |\{(x, y) \in TR \mid d(x', x) \leq \varepsilon\}| = k \quad (2.14)$$

In fase di prezione si utilizza infine una funzione di aggregazione per calcolare il vettore di output predetto. Ad esempio, si può utilizzare la media pesata in base alla distanza:

$$mean = \frac{\sum_{i=1}^k d(x^{(i)}, x') \cdot y^{(i)}}{\sum_{i=1}^k d(x^{(i)}, x')} \quad (2.15)$$

Da ciò si ottiene infine la funzione di predizione:

$$f(x') = mean(\{(x, y) \in N_k(x')\}) \quad (2.16)$$

2.2.2 Regressione Lineare

Sia data la funzione generica della regressione:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Esistono diversi tipi di regressione in base alla dimensionalità del vettore di output m e in base al tipo di funzione f .

In generale, se f è una funzione lineare, allora si parla di regressione lineare.

Regressione Lineare Semplice

Si definisce regressione lineare semplice il caso in cui la formula f è:

$$f(x) = mx + q \quad (2.17)$$

Dove m è il coefficiente angolare e q è l'intercetta. Ciò significa che la funzione f è una retta nello spazio bidimensionale.

L'obiettivo del regressore è quello di trovare i valori di m e q che minimizzano la funzione di errore.

Interpretazione Geometrica e Statistica

I valori di m e q hanno un significato specifico geometrico (figura 2.6). Infatti, m rappresenta la pendenza della retta e q rappresenta il punto di intersezione della retta con l'asse delle ordinate.

Si ha che:

- Coefficiente angolare m :
 - Valori alti di m indicano una retta con pendenza alta, quindi una retta molto inclinata
 - $m = 0$ indica che la retta è orizzontale
 - $m < 0$ corrisponde a una retta con pendenza negativa, quindi una retta orientata verso il basso

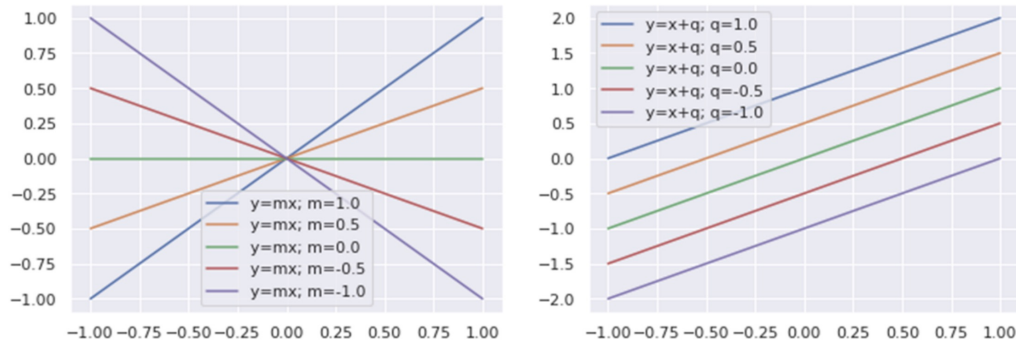


Figura 2.6: Interpretazione geometrica della regressione lineare

- Intercetta q :
 - Valori alti di q indicano che la retta interseca l'asse delle ordinate in un punto alto, ovvero che la scala dell'asse delle ordinate è alta
 - Valori bassi di q indicano che la retta interseca l'asse delle ordinate in un punto basso, ovvero una scala bassa o che i valori da predire sono negativi

A prescindere dall'interpretazione geometrica, i valori di m e q hanno anche un significato statistico, utile al fine della regressione lineare. L'intercetta infatti corrisponde al valore che verrà predetto nel caso in cui il valore di x sia 0. Questo, in base al contesto, può avere un significato specifico. Ad esempio, nel caso di un dataset che contiene il numero di ore di studio e il voto ottenuto, l'intercetta corrisponde al voto che si otterrebbe senza studiare.

Il coefficiente angolare invece corrisponde alla variazione del valore di output per ogni unità di variazione del valore di input. Questo è anche verificabile matematicamente:

$$f(x+1) - f(x) = m(x+1) + q - (mx + q) = m$$

Regressione Lineare Multipla

La generalizzazione della regressione semplice lineare è la regressione lineare multipla. In questo caso la funzione f è:

$$f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$$

e più in particolare si ottiene:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \quad (2.18)$$

$$f(x) = \theta_0 + \sum_{i=1}^n \theta_i x_i$$

In questo caso i parametri del modello sono $\Theta = \{\theta_0, \theta_1, \dots, \theta_n\}$ e la fase di training consiste quindi nel trovare i valori di Θ che minimizzano la funzione di errore.

In particolare, in questo caso specifico, la funzione costo $J(\Theta)$ è definita come lo scarto quadratico medio tra il vettore di verità y e il vettore di output predetto $\hat{y} = f(x)$:

$$J(\Theta) = \frac{1}{2} \sum_{i=1}^N (f_{\Theta}(x^{(i)}) - y^{(i)})^2 \quad (2.19)$$

Apprendimento e Discesa del Gradiente

La fase di apprendimento (training) della regressione lineare consiste nel trovare i valori di Θ che minimizzano la funzione di errore.

Definita quindi una generica funzione di errore $J(\Theta)$, l'obiettivo è quello di trovare i valori di Θ per i quali, idealmente, $J(\Theta) = 0$.

$$\Theta = \arg_{\Theta} \min J(\Theta) \quad (2.20)$$

Ovviamente calcolare i valori di $J(\Theta)$ per tutti i valori di Θ è computazionalmente impossibile. Si utilizza quindi un approccio iterativo: si sfrutta l'algoritmo della discesa del gradiente, che permette di minimizzare le fun-

zioni differenziabili.

L'algoritmo infatti si basa sull'idea secondo la quale, se la funzione costo è definita e differenziabile nell'intorno di un punto $\theta^{(0)}$, allora basterà seguire l'andamento del grafico della funzione costo stessa. Ciò significa che data la derivata della funzione costo nel punto $\theta^{(0)}$, allora la funzione costo tenderà a diminuire seguendo la direzione opposta alla derivata.

Questo perchè, da interpretazione geometrica, la derivata prima di una funzione in un punto è il coefficiente angolare della retta tangente al grafico della funzione nel punto stesso.

L'algoritmo è iterativo, l'obiettivo è infatti quello di spostarsi di volta in volta in un punto $\theta^{(i)}$ tale che $J(\theta^{(i)}) < J(\theta^{(i-1)})$, così facendo, idealmente, si arriva a un minimo globale della funzione costo.

A ogni iterazione i si calcola quindi il punto $\theta^{(i)}$ come:

$$\theta^{(i)} = \theta^{(i-1)} - \gamma \cdot J'(\theta^{(i-1)}) \quad (2.21)$$

con γ parametro di apprendimento. Riportando un esempio in figura 2.7, con punto di partenza $\theta^{(0)} = 0.4$, $\gamma = 0.02$ e $J'(\theta^{(0)}) = -1.8$, si ottiene:

$$\theta^{(1)} = 0.4 - 0.02 \cdot (-1.8) = 0.436 \quad (2.22)$$

In pratica, l'algoritmo itererà fino a quando non si raggiunge il minimo della funzione costo o finchè non si soddisfa una condizione di uscita:

- Si raggiunge il numero massimo di iterazioni.
- La differenza tra il valore della funzione costo tra due iterazioni successive è minore di una soglia ε .
- il valore di $\gamma \cdot J'(\theta^{(i)})$ è minore di una soglia ϵ .

È importante notare che, idealmente, l'algoritmo converge verso un minimo globale della funzione costo, ma non è detto che ciò accada sempre. Infatti, ciò accade solo se la funzione è convessa. Nella pratica però, sono rari i casi in cui la funzione costo è convessa. In questi casi, quindi, l'algoritmo converge semplicemente verso un minimo locale. Ciò non è necessariamente

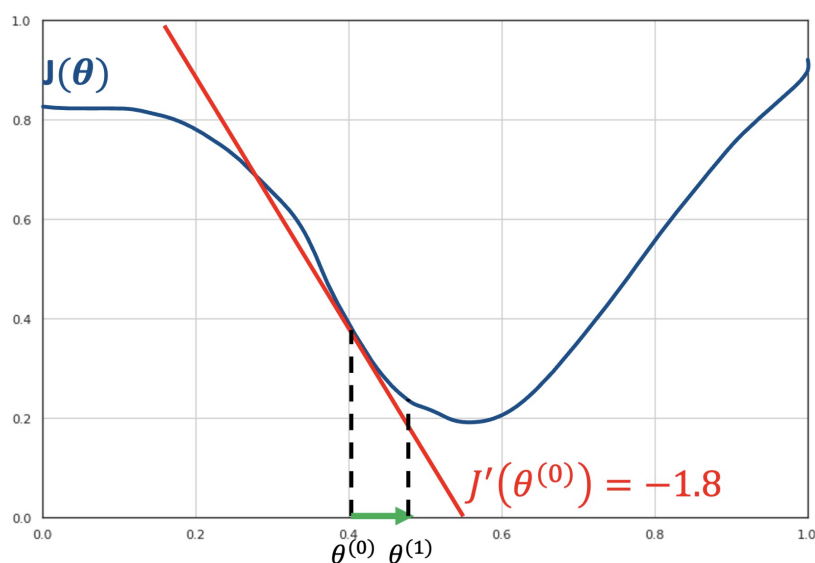


Figura 2.7: Esempio di discesa del gradiente a singolo parametro

un problema, in quanto il minimo locale offre spesso risultati accettabili.

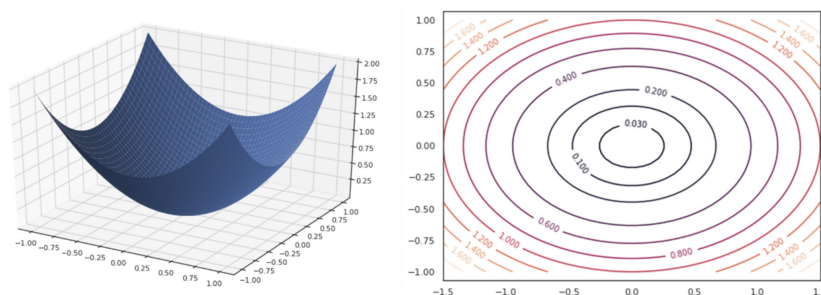


Figura 2.8: Grafico di funzione a 2 variabili

L'algoritmo della discesa del gradiente appena descritto lavora con una singola variabile θ . Nel caso della regressione lineare multipla invece si hanno più variabili $\theta_0, \theta_1, \dots, \theta_n$ ed è possibile generalizzare l'algoritmo in modo tale da lavorare con più variabili.

Infatti, se la funzione costo è definita e differenziabile nell'intorno del punto, è comunque possibile calcolare le derivate parziali della funzione costo rispetto alle variabili $\theta_0, \theta_1, \dots, \theta_n$ e generare così un vettore di derivate parziali $\nabla J(\Theta)$ che indicherà il gradiente della funzione nel punto $\Theta =$

$\{\theta_0, \theta_1, \dots, \theta_n\}$:

$$\nabla J(\Theta) = \begin{pmatrix} J_{\theta_0}(\Theta) \\ J_{\theta_1}(\Theta) \\ \vdots \\ J_{\theta_n}(\Theta) \end{pmatrix} \quad (2.23)$$

Prendendo ad esempio una funzione costo a due variabili $J(\theta_0, \theta_1)$ (figura 2.9), con le stesse ipotesi di prima, si ottiene $\nabla J(\Theta)$ che indicherà il gradiente della funzione nel punto $\Theta = \{\theta_0, \theta_1\}$:

$$\nabla J(\Theta) = \begin{pmatrix} J_{\theta_0}(\Theta) \\ J_{\theta_1}(\Theta) \end{pmatrix} \quad (2.24)$$

La modifica dei punti θ_n rimane invariata rispetto al caso a singola variabile, viene semplicemente applicata a tutte le variabili in corrispondenza con la loro derivata parziale.

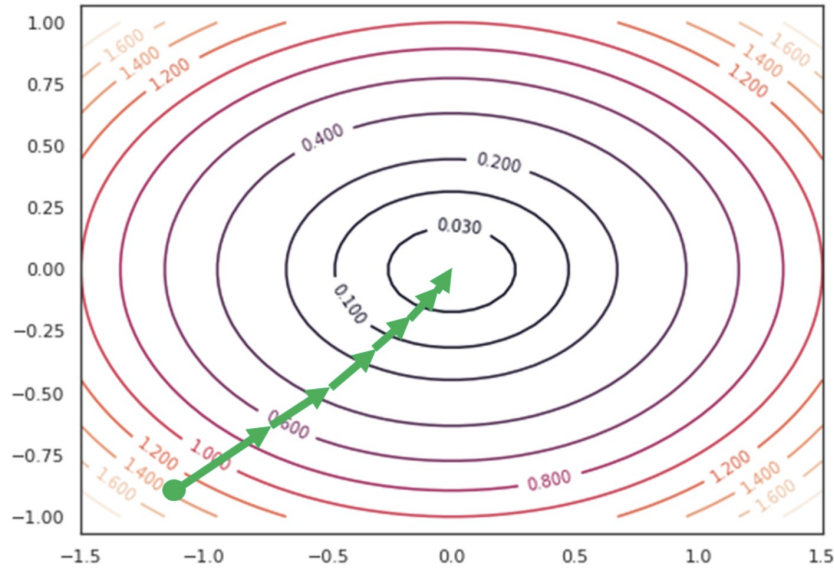


Figura 2.9: Esempio di discesa del gradiente a 2 variabili

2.2.3 Regressione Ordinale

La regressione ordinale è un problema di regressione usato per predire una variabile ordinale, ovvero una variabile che può assumere un numero finito di valori ordinati tra loro, dove quindi l'ordinamento è significativo nel peso della predizione.

È definibile come un problema intermedio tra la regressione e la classificazione. Questo perché la predizione avviene comunque su una variabile continua, ma il risultato è una classe ordinale, ovvero valori discreti ordinati tra loro. La funzione generica della regressione ordinale è:

$$f : \mathbb{R}^n \rightarrow \{1, 2, \dots, n\}$$

Un esempio di regressione ordinale è la predizione del grado di soddisfazione di un utente in base ad un prodotto, espresso ad esempio con una scala da 1 a 5.

La regressione ordinale si esegue usando un modello lineare che adatta sia un vettore di coefficienti ω , sia un insieme di soglie $\{\theta_0, \theta_1, \dots, \theta_n\}$, dove n è il numero di classi, ovvero i possibili valori di output. Si definiscono quindi i vettori di input x e le loro corrispondenti classi $y \in \{1, 2, \dots, n\}$ e tali che $y^{(i)} \leq y^{(j)}$ se $i \leq j$. L'obiettivo del regressore sarà quello di trovare i pesi giusti per i coefficienti ω e le soglie $\{\theta_0, \theta_1, \dots, \theta_n\}$ con la proprietà $\theta_0 < \theta_1 < \dots < \theta_n$ in modo tale da dividere lo spazio in n regioni, una per ogni classe.

La funzione di predizione sarà quindi:

$$f(y \leq i \mid x) = \sigma(\theta_i - \omega \cdot x) \quad (2.25)$$

Si ottiene quindi che la probabilità cumulativa della classe y di essere almeno i è data da una funzione σ applicata a una funzione lineare x . La funzione σ può essere ad esempio la funzione logistica, creando il modello logit ordinale:

$$\sigma(\theta_i - \omega \cdot x) = \frac{1}{1 + e^{-(\theta_i - \omega \cdot x)}}$$

Predizione

I modelli di regressione ordinale possono essere interpretati assumendo l'esistenza di una variabile latente y^* :

$$y^* = \omega \cdot x + \varepsilon$$

Dove ε è il termine di errore, che si presume segua una distribuzione logistica standard.

La predizione della classe y consiste semplicemente in un calcolo incompleto di y^* , dove si determina solo l'intervallo in cui y^* cade:

$$y = \begin{cases} 1 & \text{se } y^* \leq \theta_1 \\ 2 & \text{se } \theta_1 < y^* \leq \theta_2 \\ \vdots & \\ n & \text{se } \theta_{n-1} < y^* \leq \theta_n \end{cases} \quad (2.26)$$

Assumendo che $\theta_0 = -\infty$ e $\theta_n = +\infty$, (equazione 2.26) si riassume come $y = k \Leftrightarrow \theta_{k-1} < y^* \leq \theta_k$.

Utilizzando sempre la funzione logistica σ , si ottiene quindi la probabilità che $y = k$:

$$\begin{aligned} P(y = k \mid x) &= P(\theta_{k-1} < y^* \leq \theta_k \mid x) \\ &= P(\theta_{k-1} < \omega \cdot x + \varepsilon \leq \theta_k \mid x) \\ &= \sigma(\theta_k - \omega \cdot x) - \sigma(\theta_{k-1} - \omega \cdot x) \end{aligned} \quad (2.27)$$

Da ciò si deriva quindi la Log-Likelihood per un singolo input (x_i, y_i) :

$$L(\omega, \theta \mid x_i, y_i) = \sum_{n=1}^N [y_i = n] \cdot \log(\sigma(\theta_n - \omega \cdot x_i) - \sigma(\theta_{n-1} - \omega \cdot x_i)) \quad (2.28)$$

Dove $[y_i = n]$ è una funzione indicatrice che vale 1 se $y_i = n$ e 0 altrimenti. L'obiettivo è quindi quello di massimizzare la Log-Likelihood per tutti gli input (x_i, y_i) , ovvero minimizzare la Log-Likelihood negativa tramite ad

esempio la discesa del gradiente.

$$nll(\omega, \theta \mid x_i, y_i) = - \sum_{n=1}^N [y_i = n] \cdot \log(\sigma(\theta_n - \omega \cdot x_i) - \sigma(\theta_{n-1} - \omega \cdot x_i)) \quad (2.29)$$

2.3 Misure di Performance

Seguendo il paradigma del machine learning, è necessario valutare le performance del modello creato. Per fare ciò, si utilizzano funzioni specifiche con lo scopo di quantificare l'errore o l'accuratezza del modello.

Definito un dataset D già diviso nei suoi sottoinsiemi utili e osservando il test set TE , si seleziona l'insieme dei valori di verità:

$$Y_{TE} = \{y^{(i)} \mid (x^{(i)}, y^{(i)}) \in TE\}$$

Si seleziona poi l'insieme generato dai valori predetti:

$$\hat{Y}_{TE} = \{f(x^{(i)}) \mid (x^{(i)}, y^{(i)}) \in TE\}$$

Idealmente, l'obiettivo ottimale sarebbe quello di avere una corrispondenza biunivoca tra i valori di verità e i valori predetti, ovvero che $Y_{TE} = \hat{Y}_{TE}$. In pratica sarà compito della funzione di performance di valutare l'errore generato dalla predizione.

2.3.1 Root Mean Squared Error

Dato che sia Y_{TE} che \hat{Y}_{TE} sono insiemi di valori reali (\mathbb{R}^m), un primo approccio al calcolo dell'errore è quello di calcolare la distanza tra i due insiemi. Ad

esempio, utilizzando la distanza euclidea:

$$\|\hat{y} - y\|_2 = \sqrt{\sum_{i=1}^m (\hat{y}_i - y_i)^2} \quad (2.30)$$

Solitamente però si tende a elevare al quadrato la distanza euclidea per penalizzare maggiormente gli errori più grandi:

$$\|\hat{y} - y\|_2^2 = \sum_{i=1}^m (\hat{y}_i - y_i)^2 \quad (2.31)$$

Applicando ciò a tutto il test set si ottiene quindi una funzione di performance che misura l'errore medio commesso dal modello, ovvero la Mean Squared Error (MSE). Minore è il valore della MSE, migliore è il modello.

$$MSE = \frac{1}{|TE|} \sum_{j=1}^{|TE|} \|\hat{y}^{(j)} - y^{(j)}\|_2^2 \quad (2.32)$$

L'interpretazione logica della Mean Squared Error però non è immediata, questo perchè l'unità di misura dell'MSE è il quadrato dell'unità di misura della variabile di output.

Per ovviare a ciò si utilizza la Root Mean Squared Error (RMSE), che è semplicemente la radice quadrata della MSE:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{|TE|} \sum_{j=1}^{|TE|} \|\hat{y}^{(j)} - y^{(j)}\|_2^2} \quad (2.33)$$

2.3.2 Mean Absolute Error

Un'alternativa alla MSE è la Mean Absolute Error (MAE). Applicabile quando i valori di output predetti sono scalari, è semplicemente la media del valore assoluto della differenza tra il valore di verità e il valore predetto:

$$MAE = \frac{1}{|TE|} \sum_{j=1}^{|TE|} |\hat{y}^{(j)} - y^{(j)}| \quad (2.34)$$

La differenza principale è che la MAE non eleva al quadrato la differenza tra i valori, ma ne prende il valore assoluto. Questo ha come conseguenza che la MAE non penalizza maggiormente gli errori più grandi, ma li considera tutti uguali.

2.4 Word Embedding

Dato un dataset di testo, per poter applicare un generico algoritmo di machine learning, è necessario convertire le parole in input in vettori comprensibili all'algoritmo stesso.

Un semplice approccio è, ad esempio, quello di utilizzare un dizionario che associa ad ogni parola un vettore di valori binari, dove ogni valore indica la presenza o meno di una parola all'interno del testo. Questo approccio però si limita semplicemente a codificare la presenza di una parola, senza considerare il contesto in cui la parola è utilizzata: in presenza di sinonimi o parole con significati simili, il dizionario non sarebbe in grado di distinguere le parole.

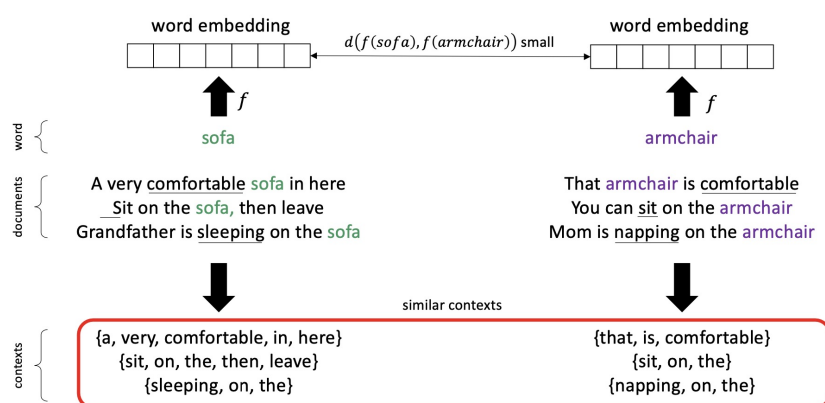


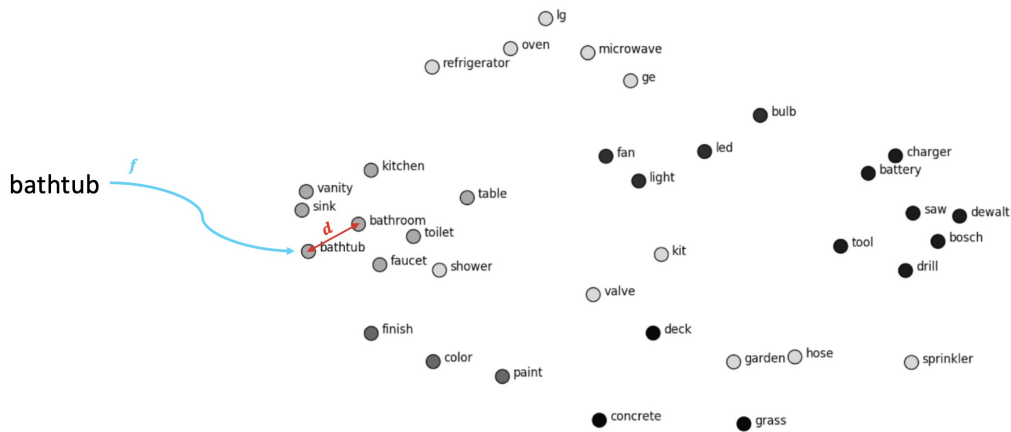
Figura 2.10: Parole con contesti simili avranno Embedding simili

È necessario quindi un approccio che codifichi la semantica delle parole: parole con significati simili devono essere codificate similmente. Tale grado di similarità deve essere misurabile, utilizzando, ad esempio, la distanza tra le parole.

L'obiettivo è quindi quello di creare uno spazio dove mappare le parole attraverso una funzione f , tale che la distanza tra due parole w_1, w_2 sia proporzionale alla loro similarità semantica.

Un approccio semplice per riconoscere la semantica di una parola è quello di considerare il contesto in cui la parola è utilizzata in quanto parole con significati simili sono spesso utilizzate in contesti simili (figura 2.10). L'estrazione del contesto può essere fatta, ad esempio, considerando le parole che compaiono in una finestra di n parole centrata nella parola in questione. Idealmente quindi, si vorrebbe una rappresentazione vettoriale a lunghezza fissa per ogni parola, tale che la distanza tra i vettori sia proporzionale alla similarità semantica tra di esse, come in figura 2.11.

Esistono diversi metodi per la generazione di questo tipo di rappresentazione vettoriale basati su diversi approcci: riduzione spaziale della matrice di co-occorrenza [4], modelli probabilistici sulla matrice di co-occorrenza [5], reti neurali [6] [7]. Quest'ultimo, utilizzando le reti Transformer, è il metodo più utilizzato e più efficace per la generazione di word embeddings.



$$d(\text{bathtub}, \text{bathroom}) = d(f(\text{bathtub}), f(\text{bathroom}))$$

Figura 2.11: Embeddings in uno spazio euclideo

2.5 Riduzione Dimensionale

La riduzione dimensionale è una tecnica di machine learning che permette di ridurre la dimensionalità di un dataset, ovvero il numero di variabili che lo compongono. Questo è utile per diversi motivi:

- Riduzione del rumore: eliminando le variabili meno significative, si riduce il rumore presente nel dataset.
- Riduzione del costo computazionale: eliminando le variabili meno significative, si riduce il costo computazionale necessario per l'addestramento del modello.
- Visualizzazione: riducendo la dimensionalità del dataset, è possibile visualizzare il dataset in uno spazio bidimensionale o tridimensionale.

In base al tipo di dati trattati e al tipo di problema, esistono diversi metodi per la riduzione dimensionale.

2.5.1 PCA - Principal Component Analysis

La Principal Component Analysis (PCA) è un metodo di riduzione dimensionale che si pone l'obiettivo di ridurre il numero di variabili che definiscono il dataset, limitando il più possibile la perdita di informazioni.

La PCA è un algoritmo non supervisionato, ovvero che non utilizza le etichette dei dati per effettuare la riduzione dimensionale. Tale riduzione avviene tramite una trasformazione lineare che mappa le variabili originali in un nuovo spazio cartesiano: qui la nuova variabile con varianza più alta viene proiettata sul primo asse, la seconda variabile con varianza più alta viene proiettata sul secondo asse e così via.

Un approccio più semplice, spesso utilizzato, sfrutta invece la matrice di covarianza del dataset. La matrice di covarianza è una matrice quadrata simmetrica che contiene le covarianze tra le variabili del dataset. In principio, l'algoritmo calcola gli autovalori della matrice di covarianza: si ottengono

tanti autovalori quante sono le variabili del dataset.

Partendo dall'autovalore più alto, si calcola l'autovettore corrispondente. Si avranno quindi tanti autovettori quante sono le righe della matrice di covarianza.

Ogni autovettore può potenzialmente essere moltiplicato per la corrispondente variabile originale x_i per ottenere la nuova variabile w_i . In pratica invece, si costruisce la matrice degli autovettori V detta di rotazione. Da tale matrice si potrà quindi calcolare la matrice W dei nuovi valori, ottenuta moltiplicando la matrice di rotazione per la matrice dei valori originali.

$$W = V \cdot X$$

Così facendo si ottengono le coordinate dei punti nel nuovo spazio.

PCA è in grado di lavorare con variabili continue e numeriche, ma non di lavorare con variabili categoriche. Per variabili di quest'ultimo genere è opportuno utilizzare altri metodi di riduzione dimensionale: MCA (Multiple Correspondence Analysis) o FAMD (Factor Analysis of Mixed Data) sono entrambe estensioni di PCA che permettono ciò.

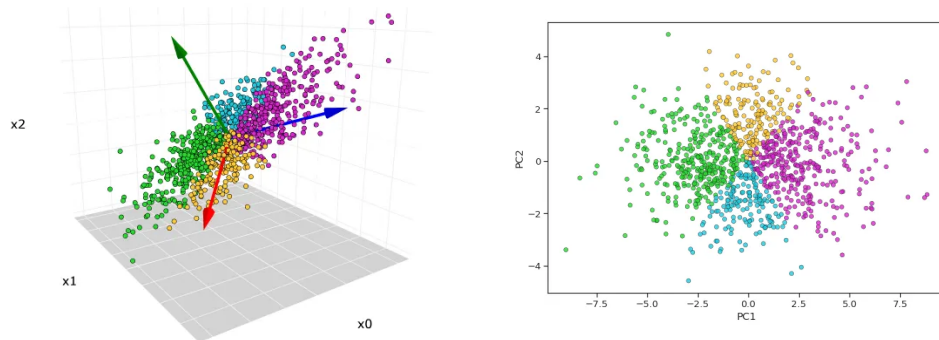


Figura 2.12: Esempio di riduzione dimensionale con PCA da 3 dimensioni (sinistra) a 2 dimensioni (destra)

2.5.2 Standard Scaler

Lo Standard Scaler non è un algoritmo di riduzione dimensionale, ma è un algoritmo di preprocessing spesso utile prima di applicare un algoritmo di riduzione dimensionale, in casi specifici anche necessario (come per PCA).

L'algoritmo si limita a trasformare i valori di un dato vettore x in modo tale che la media sia 0 e la deviazione standard sia 1. Per fare ciò si calcola la media μ e la deviazione standard σ del vettore x :

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$
$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

Il prossimo passo è quello di sottrarre ad ogni valore di x la media μ e dividere il risultato per la deviazione standard σ :

$$x' = \frac{x - \mu}{\sigma} \tag{2.35}$$

Così facendo si ottiene un nuovo vettore x' standardizzato.

Capitolo 3

Dati utilizzati

Si elencano di seguito i dati utilizzati per la realizzazione del sistema di raccomandazione.

Sono stati usati due dataset:

- Catalogo Netflix: contiene circa 6 mila film e serie TV con le relative informazioni (anno di uscita, genere, attori, regista, etc.) disponibili sulla piattaforma.
- Netflix Prize Dataset [8]: contiene 100 milioni di valutazioni di 480 mila utenti su 17 mila film. I dati sono stati raccolti tra il 1998 e il 2005.

Ognuno dei due dataset è stato utilizzato per uno scopo specifico, hanno quindi subito fasi di pre-processing differenti.

Per tutte le operazioni di pre-processing e analisi dei dati è stato utilizzato il linguaggio di programmazione Python.

3.1 Dataset

Si elencano di seguito le caratteristiche dei dataset utilizzati nel progetto.

3.1.1 Catalogo Netflix

Il dataset del catalogo Netflix è stato generato tramite scraping del sito web *movieplayer* [9]. Il sito contiene una lista di film e serie TV disponibili sulla piattaforma Netflix, con le relative informazioni, aggiornata giornalmente.

Scraping

Per effettuare lo scraping del sito è stato utilizzato il framework *Selenium*. Il sito fornisce una prima sezione che contiene semplicemente la lista degli elementi, film o serie TV che siano. Utilizzando Selenium, è stato creato uno script Python in grado di costruire e mantenere una lista dei film e delle serie TV disponibili, al momento dell'esecuzione dello script stesso. Questo elenco simula una cache, usata per registrare gli ultimi contenuti noti e inclusi nel dataset. Ad ogni esecuzione dello script, viene verificata la data di creazione della lista. Se sono trascorsi più di 7 giorni dall'ultima esecuzione, verrà effettuato un nuovo aggiornamento.

Successivamente, per ogni film o serie TV presente nella lista, viene effettuata una richiesta al sito web per spostarsi nella pagina dedicata al contenuto.

Una singola pagina contiene tutte le informazioni relative al film o alla serie TV (come in figura 3.1), in particolare si è scelto di estrarre le seguenti informazioni:

- Titolo
- Data di uscita
- Anno di uscita
- Genere
- Regista
- Attori
- Paese

Forrest Gump 1994

★ 4.0/5
★ 3.8/5
VOTA

SCHEDA

CAST

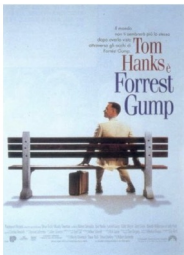
STREAMING

VIDEO

CURIOSITÀ

NEWS E ARTICOLI

PREMI



Forrest Gump è un film di genere Drammatico, Commedia del 1994 diretto da Robert Zemeckis con Tom Hanks e Robin Wright. Durata: 142 min. Paese di produzione: USA.

IN STREAMING

INDICE

- 1 Trama
- 2 Cast
- 3 Curiosità e frasi celebri
- 4 Accoglienza
- 5 Video
- 6 Immagini e foto
- 7 Home video
- 8 News e articoli

Data di uscita	6 Ottobre 1994 (Italia) 6 Luglio 1994 (USA)
Genere	Drammatico , Commedia
Anno	1994
Regia	Robert Zemeckis
Attori	Tom Hanks, Robin Wright, Gary Sinise, Mykelti Williamson, Sally Field, Haley Joel Osment, Elizabeth Hanks
Paese	USA
Durata	142 Min

TRAMA DEL FILM FORREST GUMP

Forrest Gump, affetto da un leggero deficit mentale, racconta seduto in una panchina il suo passato e il suo amore per Jenny. Forrest ha affrontato vari momenti della storia americana, compresa la guerra in Vietnam, riuscendo a cavarsela in qualsiasi situazione.

Figura 3.1: Pagina di un film

- Durata
- Trama
- Casa di produzione (se disponibile)

3.1.2 Netflix Prize Dataset

Netflix Prize è una competizione lanciata da Netflix nel 2006, con lo scopo di migliorare il sistema di raccomandazione della piattaforma.

La competizione è terminata nel 2009, con la vittoria del team *BellKor's Pragmatic Chaos* [10], che ha fornito una soluzione che migliorava del 10.06% il sistema di raccomandazione di Netflix.

L'obiettivo era quello di fornire una miglioria alla RMSE originale di almeno il 10%.

Il dataset originale è composto da 100 milioni di valutazioni di 480 mila utenti su 17 mila film, completamente anonimizzati.

Allo scopo di questo progetto, è stato utilizzato un sottoinsieme di questo dataset, contenente circa 1.7 milioni di valutazioni sui 53 film in comune con il dataset del catalogo Netflix, costruito in precedenza.

3.2 Pre-processing

Entrambi i dataset hanno subito una fase di preprocessing, proprietaria per ognuno di essi, per essere adattati alle esigenze del progetto.

3.2.1 Catalogo Netflix

Il dataset del catalogo Netflix, contenente le informazioni sui film e le serie TV, è stato utilizzato come fonte di metadata per la costruzione dei vettori identificativi dei film.

Embeddings delle trame

In principio, per ogni elemento, si è estratta la trama e, tramite l'utilizzo del transformer *multilingual-e5-large-v2* [11], si è calcolato l'embedding della stessa. La rete neurale, dato in input un testo di lunghezza massima pari a 512 token, restituisce un vettore di 1024 dimensioni.

Dato che la rete neurale accetta solamente un massimo di 512 token in input, si è scelto di stimare la lunghezza massima della trama, in termini di token, eseguendo semplicemente il conteggio delle parole. Questa stima, pessimistica dato che non tiene conto della rimozione delle stop words, permette

comunque di avere una metrica sufficientemente buona della lunghezza massima della trama.

Nel caso in cui quanto stimato superasse quindi i 512 token, si è scelto di dividere la trama per frasi: ogni frase avrà un suo vettore di embeddings e il vettore finale sarà la media dei vettori delle singole frasi.

```
1 def encode(trama, model):
2     if len(trama.split(" ")) > 512:
3         totalEmbeddings = []
4         for line in trama.split("\n"):
5             totalEmbeddings.append(model.encode("query: " +
6                 line))
7         return np.array(totalEmbeddings).mean(axis=0)
8     else:
9         return model.encode("query: " + trama)
```

Embeddings Metadata e Dimensionalità

Successivamente, si è deciso di utilizzare anche le informazioni relative al genere, agli attori e al regista. Queste, essendo informazioni categoriche, sono state codificate utilizzando l'algoritmo *one-hot encoding* o *multilabel binarizer* nel caso in cui un film o una serie TV avesse più di un genere, attore o regista.

Dopo aver creato i vettori rappresentativi per ogni feature dei film, l'obiettivo è quello di generare un unico vettore rappresentativo per ogni film, concatenando quanto calcolato fin'ora. Nel fare ciò però si presentano diversi problemi riguardo alla dimensionalità dei vettori, in particolare quelli delle feature categoriali.

Questo perchè l'algoritmo *one-hot encoding* crea un vettore di dimensione pari al numero di categorie univoche presenti nella feature.

Per evitare quindi rappresentazioni con dimensioni troppo grandi, si è scelto di concatenare tutte le feature in un unico vettore e successivamente applicare un algoritmo di riduzione della dimensionalità, in particolare *PCA*.

Prima di applicare però l'algoritmo è necessario normalizzare i dati utilizzando lo *StandardScaler*.

3.2.2 Netflix Prize Dataset

Successivamente al primo filtraggio del Netflix Prize Dataset, dove sono stati selezionati solamente i film in comune con il dataset del catalogo, si costruisce la matrice di utilità U . Questa dispone di una riga per ogni utente, identificato da un id univoco, e una colonna per ogni film, anch'essi identificati da un id.

Capitolo 4

Approccio

Il problema della raccomandazione *content-based* è stato affrontato in due modi differenti:

1. Qualitativo: si è eseguita la raccomandazione semplicemente osservando le feature dei film, senza alcun tipo di elaborazione.
2. Quantitativo: utilizzando il dataset dei ratings, si è usato un approccio che sfrutta la regressione per eseguire la raccomandazione, osservando quindi il comportamento dell'utente.

4.1 Approccio Qualitativo

Il primo approccio utilizzato è stato quello qualitativo, dove si segue il funzionamento di un sistema di raccomandazione *content-based* classico, come spiegato nel capitolo 2.1.1.

In realtà, in principio, ci si è concentrati sulla creazione di un semplice prompt da linea di comando. Esso permette di inserire il titolo di un film e, se questo dovesse essere presente nel dataset, il sistema risponderà con una lista dei 10 film più simili.

La similitudine è calcolata utilizzando la similitudine del coseno oppure, tramite scelta esplicita, la distanza euclidea. È un approccio primitivo in quanto

tiene conto semplicemente delle caratteristiche dei film, senza alcun tipo di elaborazione o filtraggio dovuto al comportamento dell'utente.

Teorico

L'approccio qualitativo principale si basa sul processo di raccomandazione *content-based* classico.

Per ogni utente si costruisce il suo vettore profilo p_u , che rappresenta le sue preferenze, seguendo l'equazione 2.3.

La raccomandazione consiste nel consigliare i primi 10 film più simili al vettore profilo dell'utente, calcolando quindi la distanza tra il vettore profilo e i vettori rappresentativi dei film (come in figura 4.1).

	HP1	HP2	HP3	TW	SW1	SW2	SW3		
A	4			5	1			HP2: [1 0 1 0 1 1]	1 $u(A, HP2) = 0.18$
B	5	5	4					HP3: [1 1 1 0 1 1]	2 $u(A, HP3) = 0.06$
C				2	4	5		SW2: [0 1 0 1 0 0]	4 $u(A, SW2) = -0.36$
D		3					3	SW3: [0 1 1 1 0 0]	3 $u(A, SW3) = -0.14$
$profile(A) = [0.77, -0.78, 0.55, -0.56, -0.01, 0]$									recommend in order

Figura 4.1: Raccomandazione Teorica puramente basata sulla similarità

4.2 Approccio Quantitativo

Il secondo approccio utilizzato è stato quello quantitativo, dove si prova a risolvere il problema della raccomandazione come un problema di regressione. Ci si concentra infatti su una scelta ristretta di misure di performance per la regressione e si cerca di minimizzare l'errore, scegliendo quindi il modello che meglio si adatta ai dati o, più in dettaglio, i parametri del modello che minimizzano l'errore.

Per il problema in questione, si sono scelte:

- RMSE
- MAE

4.2.1 Regressione

Dato che il problema che si sta affrontando è quello della raccomandazione *content-based*, ci si concentra quindi su un utente alla volta.

Data la matrice di utilità U formata da m utenti e n colonne (una per film), allora si può considerare un singolo utente u come un vettore di n elementi, dove ogni elemento è il rating dato dall'utente u al film i .

Il problema della raccomandazione, trasformato in un problema di regressione, viene gestito riga per riga, isolando quindi ogni utente e considerando solo i film che ha valutato. A ogni utente corrisponde un'istanza unica del modello scelto e si valuteranno le prestazioni della singola, mediando poi tutti i risultati ottenuti per avere una misura riassuntiva.

I modelli scelti per il problema sono:

- Regressione Lineare
- Regressione Ordinale
- K-NN Regressor

Regressione Lineare

La regressione lineare è stata scelta come modello di riferimento, data la sua semplicità e data la complessità della rappresentazione generata per i film, non ci si aspetta che il modello colga bene la relazione tra le feature e i rating. Inoltre, in una prospettiva a lungo termine, il modello lineare richiede una fase di training ogni volta che si ottiene un feedback dall'utente e si aggiunge quindi un film al training set, appesantendo quindi il sistema.

Regressione Ordinale

Un modello più complesso utilizzato è quello della regressione ordinale, che permette di trattare il problema come se fosse una via di mezzo tra la regressione e la classificazione. Per il corretto funzionamento è necessaria però la traduzione dei rating in valori ordinali. La scelta di questo modello è dovuta al fatto che, in teoria, si adatta meglio al problema in questione, anche grazie al peso che viene dato alla distanza tra i rating e alle predizioni sbagliate. È presente comunque il problema del training da eseguire ogni volta che si ottiene un feedback.

K-NN Regressor

L'ultimo modello proposto è il K-NN Regressor. Questo, come spiegato nel capitolo 2.2.1, è un modello che sfrutta una misura di distanza tra gli oggetti del training set mappati per la regressione. Dato infatti un training set, il K-NN Regressor selezionerà i k elementi più vicini, con k scelto dall'utente, e predice il rating come la media dei rating dei k elementi più vicini.

Questo modello è stato scelto per la sua semplicità e per la sua versatilità nei parametri: è stato infatti possibile testare diverse metriche di distanza e diversi pesi per la media.

Il K-NN Regressor è inoltre l'unico tra i modelli scelti a essere lazy: la sua fase di training consiste semplicemente nel memorizzare il training set, senza alcun tipo di elaborazione. Questo permette quindi di aggiungere un film al training set senza dover ricalcolare il modello, la fase di training deve comunque essere eseguita ogni volta che si ottiene un feedback, ma all'atto computazionale non pesa sul sistema.

4.2.2 Misure di Performance

Per valutare le prestazioni dei modelli, si sono scelte due misure di performance:

- RMSE
- MAE

Come spiegato nel capitolo 2.3, si è scelta la RMSE in quanto permette di penalizzare maggiormente gli errori più grandi, mentre la MAE è stata scelta per avere una misura più semplice da interpretare e più omogenea.

In particolare, dati i 53 film di tutto il dataset disponibile, si è scelto di creare il training set e il test set come segue:

1. Si sceglie come iperparametro il numero di film da inserire nel training set, k .
2. Si selezionano k film casuali dal dataset.
3. Si selezionano gli utenti che hanno valutato tutti i k film e si crea il training set TR con le loro valutazioni.
4. Il test set TE è generato invece dai $53 - k$ film rimanenti. A questi si applica poi un ulteriore filtraggio eliminando, per ogni utente, i film al quale non ha dato un rating.

Dati quindi i due set, il modello viene addestrato sul training set TR e se ne misurano poi le prestazioni calcolando RMSE e MAE sui valori del test set TE .

Correzione tramite Bias

Si è anche effettuato uno studio sulle predizioni per valutare il grado e il numero di predizioni errate, in eccesso e in difetto. Questo permette di avere una visione più dettagliata delle prestazioni del modello.

Al fine di migliorare le prestazioni, si è calcolato un *bias* da aggiungere alle predizioni, in modo da compensare gli errori effettuati. In principio, si fraziona il test set TE in modo tale da mantenere il 75% dei dati nel dataset stesso, mentre il restante 25% viene utilizzato per generare un validation set VA .

Tale *bias* viene quindi calcolato sul test set VA , come la media del vettore generato dalla concatenazione degli errori effettuati in eccesso e difetto, con i segni invertiti.

Dati quindi il validation set VA , le predizioni \hat{y} e i valori di verità y , si calcolano in principio gli insiemi delle predizioni che sono in eccesso o in difetto rispetto ai valori di verità:

$$\begin{aligned} exceeded &= \{(\hat{y}^{(i)}, y^{(i)}) \mid \hat{y}^{(i)} > y^{(i)} \in VA\} \\ defect &= \{(\hat{y}^{(i)}, y^{(i)}) \mid \hat{y}^{(i)} < y^{(i)} \in VA\} \end{aligned} \quad (4.1)$$

Si calcolano ora di quanto sono errate le predizioni. Si pone attenzione soprattutto al segno, che sarà negativo per le predizioni in eccesso e positivo per quelle in difetto:

$$errors = \{y^{(i)} - \hat{y}^{(i)} \mid (\hat{y}^{(i)}, y^{(i)}) \in (exceeded \cup defect)\} \quad (4.2)$$

Infine si calcola il *bias* come la media dei valori di *errors*:

$$bias = \frac{1}{|errors|} \sum_{i=1}^{|errors|} e_i \quad \forall e_i \in errors \quad (4.3)$$

Il *bias* viene poi aggiunto alle predizioni per compensare gli errori effettuati:

$$\hat{y}^{(i)} = \hat{y}^{(i)} + bias \quad \forall \hat{y}^{(i)} \in \hat{y} \quad (4.4)$$

Le predizioni corrette tramite *bias* vengono poi valutate con le stesse metriche di prima, RMSE e MAE, ma sui dati del validation set TE .

Schema di Esecuzione

L'esecuzione del sistema consiste quindi in uno schema specifico di passi illustrati in figura 4.2. Teoricamente, alla validazione dell'utente di una certa raccomandazione, si dovrebbe aggiungere il film al training set e ricalcolare il modello, ripetendo nuovamente lo schema descritto.

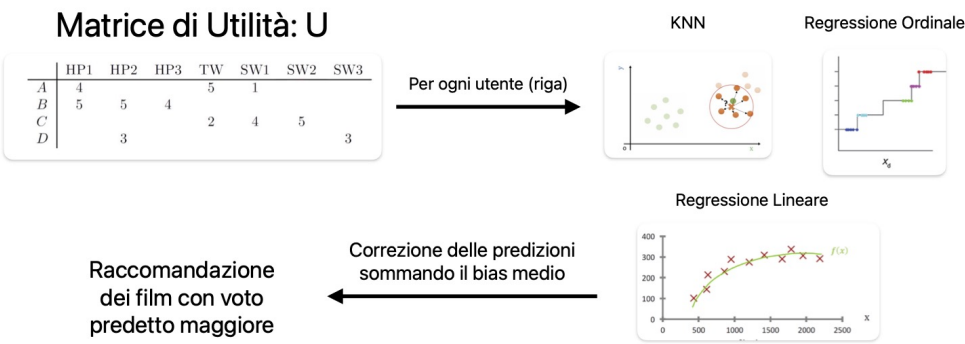


Figura 4.2: Pipeline di Esecuzione del Sistema

Capitolo 5

Esperimenti e Risultati

Nel capitolo seguente si elencano i vari esperimenti effettuati sia per valutare le prestazioni dei vari modelli, potendo quindi scegliere il migliore, sia per valutare l'efficacia dei vari parametri di configurazione al fine di massimizzare le prestazioni.

5.1 Risultati Qualitativi

Nel caso dei test qualitativi è difficile valutare esplicitamente le prestazioni dei vari modelli, in quanto non si ha un valore di riferimento per poterli confrontare in modo automatico.

A causa di ciò, le prestazioni sono state misurate manualmente, valutando la bontà delle raccomandazioni effettuate dai vari modelli.

5.1.1 Test qualitativi tra film

I primi test effettuati riguardano l'approccio qualitativo tra film. Questo, come spiegato nel capitolo 4.1, consiste nel confrontare i film del dataset con un film di riferimento, in modo da trovare i film più simili. Si mostrano i risultati della raccomandazione di film simili a "Forrest Gump" in figura 5.1.

Film più simili a 'Forrest Gump' con distanza euclidea	Film più simili a 'Forrest Gump' con similarità del coseno
Cast Away	Cast Away
Blessed Benefit	Apollo 13
Konstgjorda Svensson	Blessed Benefit
What Happened to Mr Cha?	What Happened to Mr Cha?
Dry Martina	Konstgjorda Svensson
Una vez más	L'amore all'improvviso - Larry Crowne
Perfect Strangers	Excuse Me
Excuse Me	Dry Martina
An Easy Girl	Una vez más
Chopsticks	Chopsticks

Figura 5.1: Test qualitativo tra film

Si nota come i film raccomandati siano effettivamente simili a "Forrest Gump", in quanto sono per la maggior parte film drammatici e romantici. Si nota anche che i primi due film consigliati dalla similarità del coseno siano entrambi con lo stesso attore protagonista, dettaglio che la semplice distanza euclidea sembra non cogliere del tutto.

5.1.2 Test qualitativi con profilo utente

Si valutano ora le prestazioni del sistema di raccomandazione qualitativo generato seguendo l'approccio teorico descritto nel capitolo 4.1.

Come descritto precedentemente, questo approccio utilizza già la divisione del dataset in training set e test set: questo porta a differenze di prestazioni in base al numero k di film scelti per il training set. In particolare si è notato che, per k troppo bassi, il modello non è in grado di fornire raccomandazioni di qualità, questo perchè il profilo utente non è sufficientemente rappresentativo. Per k troppo alti invece il numero di film nel test set si riduce drasticamente, rendendo complicata quindi la valutazione delle prestazioni.

I test in figura 5.2 sono stati effettuati con $k = 15$, in quanto si è notato che per valori più bassi le raccomandazioni non erano sufficientemente buone. Si nota che, tra i film del training set, siano presenti film d'azione e commedie

con rating medio-alti e queste preferenze vengono riprese nelle raccomandazioni anche se non in modo ottimale.

Film nel Profilo Utente

Titolo	Rating
Coach Carter	3
Justice League	4
National Lampoon's Vacation	3
American Psycho	4
Half Baked	3
Tales of the City	2
Fallen	4
The Bourne Supremacy	4
Ghost in the Shell	5
Disclosure	2
Training Day	4
Kickboxer	5
Scream	4
Quicksand	3
S.W.A.T.	4

Film Raccomandati

Titolo	Similarity
Vicky Cristina Barcelona	0.184479865353268
Children of the Whales	0.1821599189641731
The Land of Cards	0.1677815792007855
Hunger Games	0.1590729441717092
Sapore di te	0.1514358057431696
The Ivory Game	0.14796175600117248
Daitarn 3	0.13457649752766712
Papà cercasi	0.07506655238156895
Storia di un uomo d'azione	0.06727547370164029
Ghost Hunting	0.04251770169734275

Figura 5.2: Test qualitativo con profilo utente

5.2 Risultati Quantitativi

I test quantitativi, al contrario di quelli qualitativi, permettono di valutare le prestazioni dei vari modelli in modo diretto e automatico, in quanto si ha

un valore di riferimento per poterli confrontare nel test set.

Come spiegato nel capitolo 4.2.2, le metriche di performance scelte sono RMSE e MAE e il dataset è stato diviso in training set e test set in modo tale da massimizzare la quantità di dati validi nel test set.

I modelli scelti per i test sono i seguenti:

- Linear Regression
- Ordinal Regression
- K-NN Regressor

Il progetto si concentra principalmente sul K-NN-Regressor in quanto è l'algoritmo che modella meglio il problema, che ha più parametri da configurare e che permette di aggiungere film al training set senza dover ricalcolare il modello.

5.2.1 K-NN Regressor

Il K-NN Regressor ha diversi parametri da configurare, al fine di ottimizzare le sue prestazioni. In particolare, si ha il parametro k , che indica il numero di vicini da considerare e la metrica da utilizzare per calcolare la distanza tra i valori.

L'algoritmo permette anche di scegliere se pesare i risultati in base alla distanza: in questo modo, è più logico pesare di più i film più vicini e pesare di meno i film lontani, ma che rientrano comunque nel range del parametro k . Per tale motivo, questa tecnica è stata sempre utilizzata nei test. Infatti, idealmente si vuole che nel range del parametro k ci siano solo film simili: l'utilizzo del peso permette di avere un risultato più accurato che mitiga quindi l'errore che si genera dai film meno simili.

Parametri di configurazione

Si riportano in tabella 5.1 i risultati dei test effettuati per valutare le prestazioni del K-NN Regressor in base ai parametri di configurazione.

Tutti i test sono stati effettuati con $k = |TR|$ ovvero il numero di film nel training set, che corrisponde inoltre al massimo k possibile.

Tabella 5.1: Prestazioni dei vari parametri del K-NN misurate con RMSE

RMSE	Cosine		Euclidean	
N° Film	No Bias	Bias	No Bias	Bias
5	1.08	0.98	1.06	0.98
10	1.14	1.07	1.12	1.05
15	1.11	1.05	1.11	1.05

MAE	Cosine		Euclidean	
N° Film	No Bias	Bias	No Bias	Bias
5	0.88	0.81	0.87	0.82
10	0.98	0.93	0.97	0.91
15	0.97	0.90	0.97	0.90

Dalla tabella si nota come la metrica euclidea sia leggermente migliore della metrica del coseno, in quanto ha un RMSE e un MAE leggermente più bassi, anche se la differenza è minima.

In tutti i casi invece l'utilizzo del bias migliora le prestazioni, in quanto permette di mitigare l'errore dovuto alla media dei rating.

Si nota inoltre come anche nel caso di k bassi, il modello sia capace di mantenere costanza nelle prestazioni, in quanto non si ha un miglioramento significativo aumentando il numero di film nel training set.

Scelta del parametro K

Il parametro k è il parametro più importante del K-NN Regressor, in quanto indica il numero di vicini da considerare e pesa quindi direttamente sulle predizioni del modello.

È stata quindi effettuata una ricerca esaustiva del parametro k eseguendo, per ogni possibile dimensione del training set, un test per ogni valore di k compreso tra 2 e $|TR|$. In figura 5.3 si riportano i valori del k che ha portato al miglior risultato per ogni dimensione del training set.

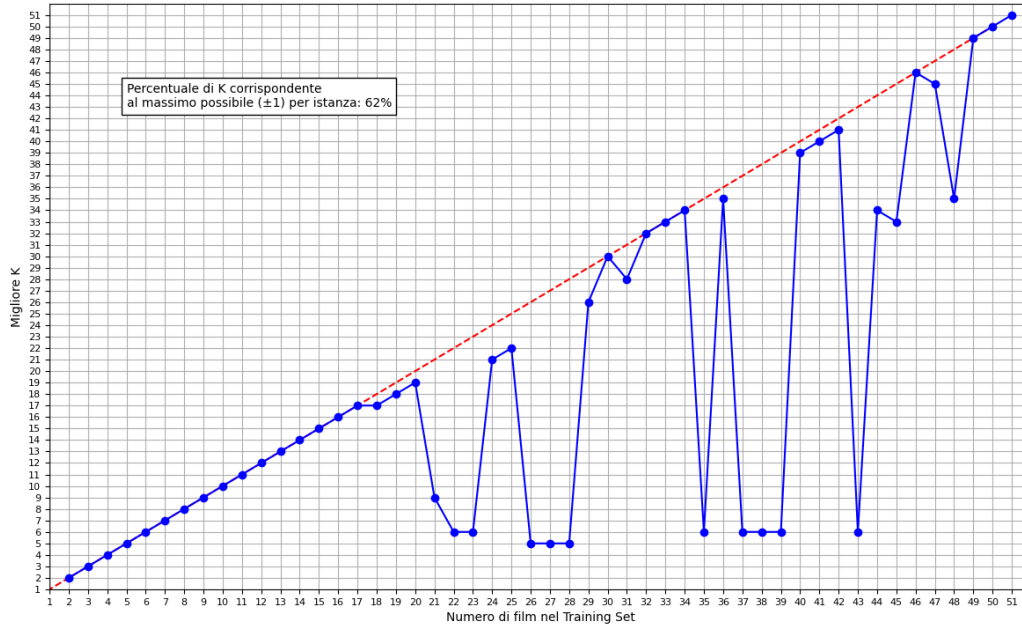


Figura 5.3: Prestazioni del K-NN Regressor in base al parametro k

Come riportato nel grafico in figura, si nota che nel 62% dei casi il miglior valore di k corrisponde alla cardinalità del training set, ovvero $k = |TR|$, con una tolleranza di ± 1 (si considera solo il caso -1 in quanto non è possibile avere $k > |TR|$). Si conteggiano pure i $k = |TR| - 1$ in quanto si è osservato che la differenza in prestazioni è minima in relazione alla possibilità di scegliere automaticamente k senza doverlo configurare manualmente. La scelta di $k = |TR|$ è quindi mediamente la migliore.

È importante anche specificare che, data la metodologia utilizzata per la costruzione del test set TE (utilizzato per la valutazione delle prestazioni), al crescere della dimensione del training set, il numero di film nel test set diminuisce, rendendo quindi più difficile la valutazione.

5.2.2 Confronto tra modelli

Si riportano di seguito i risultati dei test effettuati per confrontare i vari modelli sia con RMSE che MAE. Tutti i test sono stati eseguiti con $n = 5$ numero di film nel training set e nel caso del K-NN Regressor con $k = |TR|$,

oltre a utilizzare la metrica euclidea.

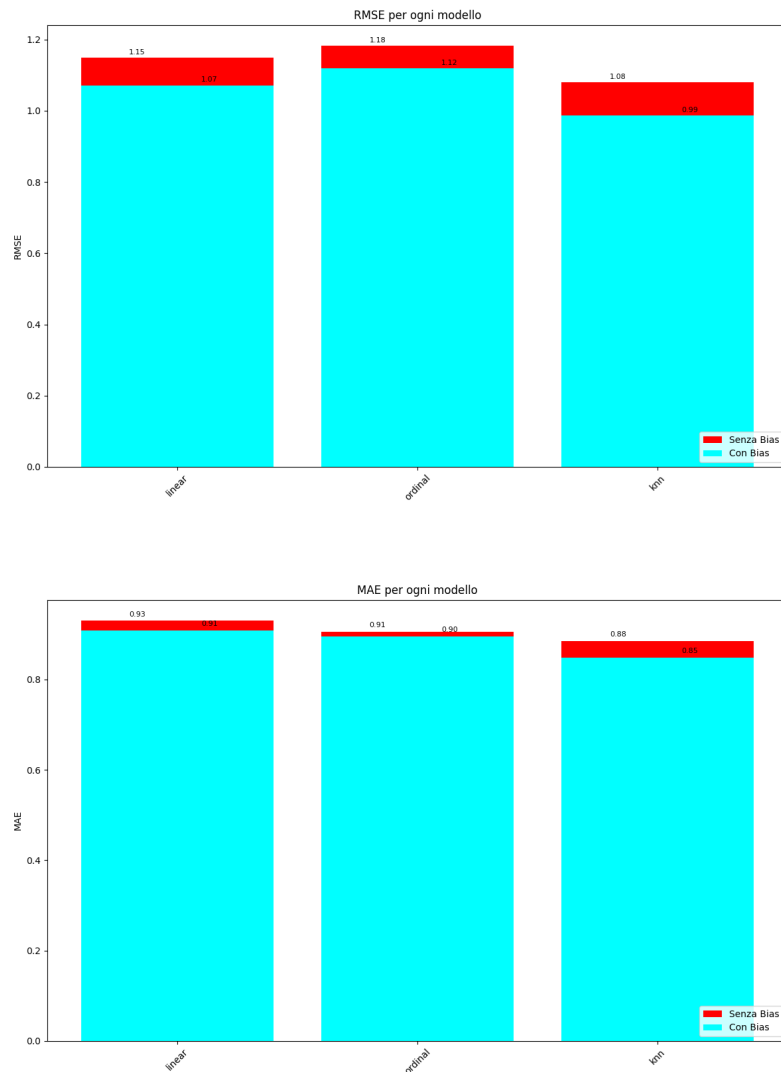


Figura 5.4: Confronto tra modelli, RMSE (superiore) e MAE (inferiore)

Si nota come il K-NN Regressor sia il modello che ottiene le prestazioni migliori, in quanto ha sia un RMSE che un MAE più bassi rispetto agli altri modelli.

Caso interessante è quello della regressione ordinale che genera un RMSE

peggiore della regressione lineare semplice ma un MAE migliore. Questo suggerisce quindi che gli errori che la regressione ordinale commette sono più gravi rispetto a quelli della regressione lineare semplice, ma sono meno frequenti.

Concentrandosi invece sull'utilizzo del bias, si nota come questo migliori le prestazioni di tutti i modelli, anche se con pesi diversi. Per i modelli lineari la differenza è di ~ 0.06 sul RMSE, mentre il K-NN Regressor è il modello che ne trae più vantaggio migliorando, con più costanza, di 0.1 sul RMSE.

Conclusione

L'obiettivo del progetto è quello di realizzare un sistema di raccomandazione content based per la piattaforma di streaming Netflix, che sia in grado di suggerire all'utente film e serie tv senza essere influenzato da fattori esterni come la popolarità del contenuto, il voto degli altri utenti o da fattori pubblicitari e di marketing.

L'obiettivo è stato anche quello di inserire nel progetto una componente di apprendimento automatico, in modo da proporre una soluzione differente da quelle già proposte per il problema. Si è trattata infatti la raccomandazione come un problema di regressione, utilizzando diversi algoritmi per la predizione dei voti dell'utente.

Tutti i modelli testati hanno lavorato con una rappresentazione dei dati basata su vettori di valori continui rappresentativi delle caratteristiche dei contenuti. Alcune di queste caratteristiche provengono da reti neurali pre-addestrate, altre sono state semplicemente codificate in valori numerici: il tutto è stato poi unito in un unico vettore.

Con questa rappresentazione, il modello che ha ottenuto i risultati migliori è stato il K-NN Regressor, che è stato in grado di predire i voti dell'utente, attraverso varie ottimizzazioni, con un RMSE di 0.9: un risultato non ottimale ma comunque accettabile su una scala di valutazione da 1 a 5.

Si nota inoltre che la soluzione proposta è capace di mantenere le stesse prestazioni anche in situazioni poco favorevoli, come nel caso di training set con pochi dati, dove quindi le informazioni sulle preferenze dell'utente sono limitate.

È giusto notare che la rappresentazione dei dati utilizzata non è ottimale per il problema, in quanto le fasi di riduzione dimensionale sono state effettuate senza un'analisi approfondita dei dati e senza un'ottimizzazione dei parametri. Una soluzione migliore sarebbe stata quella di utilizzare una rete neurale encoder-decoder, allenata con i dati originali come vettore di verità e utilizzando l'output del layer encoder come vettore con dimensione ridotta.

Bibliografia

- [1] Netflix raccomandation system. <https://research.netflix.com/research-area/recommendations#>.
- [2] Recognize strategic opportunities with long-tail data. <https://www.nngroup.com/articles/long-tail/>.
- [3] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.
- [4] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27, 2014.
- [5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [8] Netflixprize. <https://web.archive.org/web/20090924184639/http://www.netflixprize.com/>co

- [9] Film e serie tv in streaming. <https://movieplayer.it/film/netflix/>.
- [10] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10, 2009.
- [11] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.
- [12] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*, chapter 9. Cambridge University Press.
- [13] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion Books, New York, 2006.
- [14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [15] Christopher Winship and Robert D Mare. Regression models with ordinal variables. *American sociological review*, pages 512–525, 1984.
- [16] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.