Università degli Studi di Modena e Reggio Emilia

DIPARTIMENTO DI SCIENZE FISICHE, INFORMATICHE E MATEMATICHE
Corso di Laurea Magistrale in Matematica

Tesi di laurea magistrale

Confronto di modelli per il Learning To Rank: aspetti computazionali e applicazione ad un problema di ranking di documenti

Relatore:
Ch.ma Prof.ssa Silvia Bonettini

Correlatore:
Giovanni Davoli

Laureando:
Fabio Ferrari

Anno Accademico 2021-2022

«Il sapere rende liberi e l'ignoranza prigionieri»
Socrate

Sommario

Con l'avvento di Internet e un mondo orientato sempre più alla digitalizzazione, la quantità di risorse accessibili da ciascun individuo è diventata enorme. Ciò ha spinto ad un lavoro di studio e sviluppo di sistemi di Information Retrieval sempre più efficienti e in grado di rispondere con massima precisione alle svariate richieste di *sapere*. In particolare, grazie alla creazione di sistemi dotati di una capacità computazionale sempre crescente, si è cominciato ad analizzare dettagliatamente come il machine learning e l'utilizzo delle macchine possano essere applicati anche in questo ambito. È nata così una disciplina denominata *Learning to Rank*, che si propone di risolvere il problema del ranking sfruttando l'apprendimento automatico.
Il problema da risolvere, quindi, consiste nel riuscire ad ordinare in maniera opportuna le fonti di informazione che si hanno a disposizione, in modo tale che la risposta sia rilevante rispetto all'interrogazione ricevuta. L'enorme quantità di dati che si hanno a disposizione rappresenta da una parte un problema, perché i documenti più rilevanti devono essere selezionati all'interno di un vastissimo insieme, ma dall'altra consentono di addestrare adeguatamente modelli di machine learning per eseguire tale operazione nel modo più efficiente e rapido possibile. L'utilizzo di questi sistemi di calcolo ha consentito di sviluppare innumerevoli strategie in grado di adattarsi alle applicazioni più disparate.
Il learning to rank, infatti, si trova alla base di numerosissimi strumenti che tutti gli individui utilizzano nella vita di tutti i giorni. Esempi di queste applicazioni nelle esperienze quotidiane spaziano dai sistemi di raccomandazione, come quelli di Amazon, ai comuni motori di ricerca tipo Google, che devono essere in grado di stabilire quali fra tutti i documenti a disposizione devono essere forniti individuandone anche l'ordine corretto di restituzione, per evitare che l'informazione richiesta si perda tra risultati irrilevanti. In realtà queste sono solamente le situazioni più comuni, ma il learning to rank è utilizzabile anche in applicazioni differenti, come possono essere l'ambito medico, il mondo sportivo o anche il campo finanziario.
Proprio la capacità di adattamento a innumerevoli aspetti della vita quotidiana ha fatto sì che questa disciplina in breve tempo assumesse un ruolo di rilievo nell'ambiente matematico, diventando oggetto di un rapido sviluppo tuttora in corso. L'efficienza richiesta, infatti, non può essere raggiunta in altri modi se non approfondendo la matematica che giustifica le scelte implementative degli algoritmi proposti per risolvere il problema.

La presentazione e lo studio del learning to rank sarà l'argomento centrale di questo elaborato e verrà descritto in tre capitoli. Il primo è costituito da una breve spiegazione riguardante gli aspetti generali che costituiscono il machine learning e la struttura matematica posta alla base del learning to rank. In particolare, tra i possibili modelli esistenti l'attenzione è incentrata sulle rete neurali, un modello recente di intelligenza artificiale

strutturato per apprendere e risolvere i problemi imparando dall'esperienza, esattamente come farebbe l'essere umano, ma con una potenza di calcolo notevolmente maggiore.
Il secondo capitolo descrive la teoria alla base del learning to rank: dopo aver presentato i primi modelli utilizzati nell'information retrieval, verranno analizzati i tre approcci principali - *Pointwise*, *Pairwise* e *Listwise* - con cui è possibile risolvere un problema di learning to rank. Per ognuno di essi saranno anche analizzate le implementazioni di comuni algoritmi esistenti, così da poter evidenziare le principali differenze che contraddistinguono un metodo dall'altro.
Infine, nell'ultimo capitolo, dopo aver descritto in maniera più dettagliata come applicare il learning to rank alla risoluzione di differenti applicazioni reali, verrà approfondito un problema di ranking di documenti. A tal riguardo sarà utilizzato un dataset di riferimento, l'MSLR-WEB10K, così da poter confrontare i risultati ottenuti con quelli presenti in letteratura. Nel progetto presentato verrà descritto come poter implementare su Python un algoritmo per ognuno dei metodi descritti in precedenza in modo che, mediante l'utilizzo di una rete neurale, venga simulato l'operato di un motore di ricerca. In questo modo, oltre che al confronto con lo stato dell'arte esistente, sarà possibile analizzare gli aspetti computazionali relativi a ciascun algoritmo per verificare se le conclusioni teoriche proposte nel secondo capitolo siano confermate o confutate dall'implementazione pratica.

Indice

Capitolo 1

Cenni al Machine Learning e alle Reti Neurali

Per poter comprendere in maniera esaustiva gli argomenti riguardanti l'Information Retrieval (IR) e il Learning to Rank (LtR) ed il relativo progetto realizzato, è necessario possedere un'adeguata conoscenza di alcuni temi preliminari, come il machine learning e le reti neurali. Per tale ragione, in questo breve capitolo, verranno fornite le nozioni basilari essenziali per cogliere i dettagli più rilevanti della ricerca effettuata.

1.1 Machine Learning

Il *Machine Learning* (ML), che spesso è erroneamente identificato con l'*Intelligenza Artificiale* (IA), è una branca di quest'ultima disciplina scientifica sviluppatasi a partire dalla seconda metà del XX secolo. Il termine intelligenza artificiale, infatti, indica l'ambito di ricerca che ha lo scopo di sviluppare macchine in grado di prendere autonomamente decisioni, al fine di riprodurre in esse un funzionamento simile a quello ottenibile con il cervello umano. Il machine learning, invece, come suggerisce anche la sua traduzione italiana, cioè apprendimento automatico, riguarda lo studio di quegli algoritmi che consentono ai computer di apprendere informazioni dai dati e migliorare la propria prestazione nel tempo, in maniera autonoma, attraverso l'esperienza, esattamente come il cervello umano.

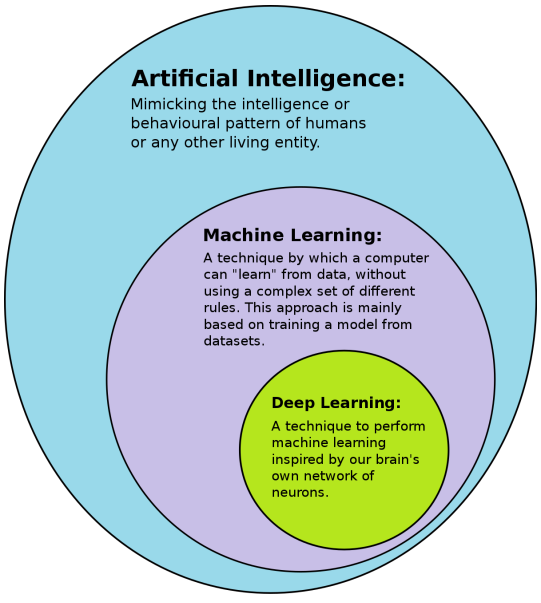


Figura 1.1: Diagramma di Eulero-Venn relativo all'intelligenza artificiale

Non esiste una definizione univoca di apprendimento automatico, ma la più utilizzata è quella di Tom M. Mitchell, scienziato informatico autore del libro intitolato appunto "Machine Learning": *Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misura-*

zione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E. Questa definizione è molto utilizzata poiché è operativa, infatti spiega come una macchina apprenda solo se c'è stato un miglioramento. Quindi, ciò che la macchina deve essere in grado di fare è, dopo aver ricevuto in input abbastanza esempi relativi al fenomeno studiato, formulare un modello che colga le proprietà generali dei dati osservati per poter poi dare in output predizioni riguardo a nuovi esempi ancora mai visti dalla macchina, ma sempre relativi allo stesso fenomeno.
Analizzando più nel dettaglio il funzionamento di un algoritmo di machine learning, serve innanzitutto che vengano raccolti abbastanza ed adeguati esempi. Infatti è molto importante che la quantità di esempi forniti sia elevata, perché ovviamente tanti più la macchina ne ha a disposizione, quanto più può imparare. Un altro aspetto fondamentale, che è forse quello principale, concerne la rappresentatività degli esempi: questa riguarda sia la classe a cui appartengono, sia la varietà globale delle singole classi che il modello dovrà distinguere, altrimenti esso potrà contare sì su tante informazioni, ma che in realtà riflettono sempre gli stessi dettagli, tralasciandone così altri, più o meno importanti che siano.
A questo punto, dopo aver scelto come rappresentare gli esempi affinché la macchina sia in grado di ricevere adeguatamente le informazioni (si pensi ad esempio alle immagini, che possono essere rappresentate sia mediante matrici di pixel sia tramite la grafica vettoriale), al computer viene passato il dataset. Al suo interno ci saranno un insieme di esempi e uno di etichette. Gli esempi, descritti attraverso d features, cioè le caratteristiche mediante le quali essi sono rappresentati, costituiranno uno spazio X d-dimensionale, perciò $X \subseteq \mathbb{R}^d$, mentre le etichette, che identificano la classe a cui ogni esempio appartiene, nel caso tradizionale di problema a singolo output, saranno monodimensionali, costituendo così lo spazio $Y \subseteq \mathbb{R}$.
Un dataset così formato è solitamente suddiviso in 3 sottogruppi: il *training set*, il *validation set* e il *test set*. Il primo contiene la maggior parte delle coppie esempio-etichetta e costituisce l'insieme dei dati, già classificati, di cui la macchina dispone per poter apprendere il modello da utilizzare nelle predizioni. Il validation set, invece, interviene nella fase di settaggio dei parametri del modello: in questa fase alla macchina, che ha già sviluppato un modello provvisorio basato sul training set, vengono passati esempi non ancora osservati su cui fare predizioni. Queste verranno confrontate con le relative etichette, che sono ancora una volta disponibili, così da capire quanti errori siano stati commessi e se sia necessario cambiare alcuni parametri nel modello per ottenere risultati più accurati. Infine, dopo aver creato il modello finale, esso viene messo alla prova mediante il test set. Di questo insieme alla macchina vengono passati gli esempi senza etichetta, in modo tale da simulare un utilizzo reale del modello, che infatti dovrà effettuare predizioni su dati mai visti prima e di cui non si conosce il risultato corretto, anche perché altrimenti sarebbe inutile utilizzare tale sistema per avere informazioni su fatti che già si conoscono. Per quanto riguarda questo ultimo insieme, in realtà, si dispongono delle etichette reali e questo permette di fare un ultimo confronto tra predizioni e valori effettivi in modo da verificare la bontà del modello creato prima del suo reale utilizzo.
Nel machine learning, la struttura matematica che sta alla base del modello è di natura probabilistica e non deterministica. Questo significa che la relazione esempio-etichetta non è certa, ma rappresenta la probabilità che un dato esempio appartenga ad una certa classe. Così facendo, in assenza di certezza, si può dire che uno stesso esempio appartiene a più di una classe, eventualmente con probabilità diverse a seconda del grado di confidenza che si ha per una etichetta piuttosto che per un'altra. Da un punto di vista formale si assume,

pertanto, che esista una certa distribuzione di probabilità $p_{xy}(x, y)$, cioè una probabilità congiunta della coppia (x, y). Sotto condizioni abbastanza generali, questa può essere riscritta come $p_{xy}(x, y) = p_x(x)p_y(y|x)$, cioè il prodotto tra la probabilità marginale $p_x(x)$, che indica la probabilità che un vettore x con quelle componenti esista, e la probabilità condizionale $p_y(y|x)$, che rappresenta la probabilità di avere una certa etichetta y dato il vettore x. Il training set, quindi, si otterrà campionando l volte lo spazio prodotto $X \times Y$ secondo la probabilità $p_{xy}(x, y)$, ottenendo l'insieme $S = \{(x_i, y_i) \in X \times Y | i = 1, \ldots, l\}$. Allora, fissato il training set S, il problema dell'apprendimento automatico si riduce alla costruzione di una certa funzione stimatrice $f_s : X \to Y$, scelta come il modello migliore all'interno di un certo spazio di funzioni prestabilito, detto spazio delle ipotesi, tale che essa sia in grado, dato un qualsiasi esempio $x \in X$, di predire il valore della $y \in Y$, cioè tale che $f_s(x) \sim y$.
In base alla struttura del training set il machine learning si suddivide in *apprendimento supervisionato* e *apprendimento non supervisionato*. Il primo caso è quello che abbiamo descritto fino ad ora, dove gli esempi vengono dati insieme alle loro etichette grazie al lavoro svolto da un etichettatore esterno, che spesso è umano. A sua volta, è possibile fare un'ulteriore distinzione per il caso supervisionato in base al range di valori che le etichette possono assumere. Si parla di *classificazione* se il numero di etichette tra cui si può scegliere è finito, indipendentemente dal nome che esse assumono. In particolare, se le etichette possibili sono solamente due, si avrà la *classificazione binaria* (è il caso dei software per il riconoscimento dell'autenticità o falsità di una banconota), mentre se gli output possibili sono più di due, ma comunque in numero finito, allora si parla di *classificazione multipla* (un esempio noto costituito da 10 classi è il caso dell'MNIST, un dataset che contiene immagini di cifre scritte a mano per la progettazione di software adibiti al riconoscimento di caratteri visivi). Invece, se $Y = I \subseteq \mathbb{R}$, cioè se le etichette assumono valori reali su tutto $\mathbb{R}$ o un suo sottoinsieme, si parla di *regressione* (come nel caso della predizione del valore di un immobile). Al contrario, nell'apprendimento non supervisionato alla macchina vengono dati in input solamente vettori x senza le etichette y. Ovviamente, in questo caso lo scopo non sarà quello di fare predizioni, non conoscendo le classi a cui attribuire gli esempi, ma gli obiettivi possono essere il *clustering*, che consiste nell'individuare pattern tra i vettori solo in base alle caratteristiche fornite, al fine di suddividere gli oggetti in gruppi per similarità (ad esempio per etichettare più facilmente i dati), o la *dimensionality reduction*, con la quale si intende l'indagine volta a capire se tutte le features siano significative ed, eventualmente, quali possano essere eliminate o aggregate mantenendo una distribuzione pressoché analoga del dataset, così da ridurre la memoria utilizzata e migliorare il costo computazionale degli algoritmi successivi basati su tale dataset.
Esistono tecniche parametriche per la creazione di algoritmi di machine learning, di cui un esempio è l'approccio di massima verosimiglianza, che cercano di parametrizzare la distribuzione di probabilità degli esempi, che è a priori sconosciuta, così da utilizzare i dati forniti per calcolare proprio tali parametri e poter così estendere lo schema anche a nuovi campioni. Tuttavia, le tecniche più utilizzate sono quelle non parametriche. Quest'ultime non si basano sulle leggi che governano la densità di probabilità dello spazio degli esempi, ma cercano di costruire direttamente il predittore mediante altre strade. L'approccio più utilizzato, che sarà infatti quello impiegato successivamente nel progetto che verrà analizzato, si incentra nell'utilizzo di una funzione loss che faccia da criterio al modello per valutare le proprie prestazioni.

1.2 Funzioni Loss

Definizione 1.1. Detta $(x, y, f(x))$ la tripla costituita da un input $x \in X$, una classe $y \in Y$ ed una predizione $f(x) \in Y$ relativa allo stesso esempio x, una funzione $V(x, y, f(x)) : X \times Y \times Y \to [0, +\infty)$ a valori reali positivi tale che $V(x, y, y) = 0 \quad \forall x \in X, \forall y \in Y$, è detta *loss-function* (funzione di costo).

In sostanza, il compito di una funzione loss è la penalizzazione degli errori di predizione fatti dal modello f. Perciò, per potersi definire funzione loss a tutti gli effetti, il comportamento necessario che essa deve avere è attribuire uno 0 quando la predizione è esatta, indipendentemente dall'esempio x su cui la predizione è stata realizzata, ed assumere altrimenti un valore reale positivo o nullo, a seconda di come si decide di implementare la funzione costo specifica. Esistono infatti vari esempi di funzioni loss di diversa complessità, come la funzione che conta gli errori, espressa da

$$V(x, y, y) = 0 \text{ se } y = f(x), \ \ 1 \text{ altrimenti,} \tag{1.1}$$

quella che attribuisce penalizzazioni diverse $\tilde{V}(x)$ a seconda dell'esempio x (perché in ambito medico, ad esempio, è più grave dichiarare sano un paziente malato piuttosto che il contrario), o quelle che si occupano di fornire predizioni non solo corrette, ma anche robuste, penalizzando pure gli output corretti che si trovano entro una zona di rischio, cioè il cui livello di confidenza è basso (una loss che utilizza questo approccio è la *soft margin loss*, utilizzata molto nelle SVM, che è definita da

$$V(x, y, f(x)) = \max(0, 1 - yf(x)) \tag{1.2}$$

con $y \in \{-1, 1\}$ e $f(x) \in [-1, 1]$). In ambito regressione, avendo output reali che difficilmente combaceranno esattamente con la y, esistono loss che analizzano la differenza tra predizione ed etichetta e attribuiscono 0 fintanto che essa è compresa in un certo intervallo di tolleranza. Un esempio è dato dalla ϵ-*insensitive loss function* che, posto $\xi = f(x) - y$, è definita come

$$\tilde{V}(\xi) = \max(|\xi| - \epsilon, 0) \tag{1.3}$$

Quelli riportati sono solamente pochissimi esempi delle loss esistenti: il comportamento delle funzioni che, in seguito, verranno utilizzate per la realizzazione del progetto sarà approfondito nei capitoli successivi.
Come si è potuto osservare sia dalla definizione che dagli esempi, una funzione loss ha il compito di valutare la performance della funzione predittrice f su un singolo esempio. È necessario, pertanto, estendere il criterio qualitativo per la f a tutti gli esempi considerati. Per farlo, si introduce il concetto di rischio atteso.

Definizione 1.2. Sia $V(x, y, f(x))$ una fissata funzione loss che misura l'errore commesso predicendo l'output di $f(x)$ sull'esempio x che ha etichetta y. Si definisce allora *rischio atteso* di f la quantità

$$R[f] := \mathbb{E}[V(x, y, f(x))] = \int_{X \times Y} V(x, y, f(x)) p_{XY}(x, y)\, dx\, dy \tag{1.4}$$

L'idea che sta alla base del rischio atteso è l'attribuzione di un numero positivo al predittore f ottenuto sommando, sull'intero spazio di cui il dataset è un sottoinsieme,

il valore della funzione loss di un esempio moltiplicato per la probabilità che quel certo esempio compaia, in quanto non è detto che ogni coppia input-output si manifesti con la stessa probabilità. Pertanto, essendo una somma di costi, maggiore sarà questo numero, peggiore sarà stato il comportamento del predittore. Quindi, scelto lo spazio delle ipotesi H entro cui scegliere il modello, colui che riesce a minimizzare il rischio atteso è il candidato perfetto, in quanto esso sarà riuscito a generalizzare in maniera migliore per predire qualsiasi possibile esempio.
Il problema è che la probabilità p_{xy} di esistere di ogni coppia non può essere ottenuta, in quanto il training set è costituito necessariamente da un numero finito di esempi. Perciò, diventa possibile calcolare solamente un'approssimazione del rischio atteso, cioè la probabilità che non si ha a disposizione, con ciò che si può ricavare dal training set, ottenendo così il rischio empirico.

Definizione 1.3. Definita la densità empirica $p_{emp}(x,y) := \frac{1}{l}\sum_{i=1}^{l}\delta_{x_i}(x)\delta_{y_i}(y)$, dove $\delta_{\xi'}(\xi)$ è la distribuzione delta di Dirac che soddisfa $\int \delta_{\xi'} f(\xi)\,d\xi = f(\xi')$, il *rischio empirico* è il funzionale definito da

$$R_{emp}[f] := \int_{X\times Y} V(x,y,f(x))p_{emp}(x,y)\,dx\,dy = \frac{1}{l}\sum_{i=1}^{l} V(x_i,y_i,f(x_i)) \tag{1.5}$$

Grazie all'introduzione di questa nuova quantità, della quale si può calcolare il minimo rispetto a f, si potrebbe ritenere che, in generale, un buon predittore sia quello che minimizza il rischio empirico. Ma, essendo quest'ultimo solo un'approssimazione del rischio atteso, per la legge dei grandi numeri questa asserzione è vera solo se il numero l di esempi del training set tende ad infinito, in quanto il teorema stabilisce che, con convergenza in probabilità, $\lim_{l\to\infty} R_{emp}[f] = R[f]$. Quindi, l'approccio del principio di minimizzazione del rischio empirico, perché possa essere utilizzato, deve essere consistente.

Definizione 1.4. Fissato uno spazio delle ipotesi H, un training set S di l elementi e denotati al suo interno i minimizzatori del rischio atteso e del empirico con, rispettivamente, $f_H = \arg\min_{f\in H} R[f]$ e $\hat{f}_{H,l} = \arg\min_{f\in H} R_{emp}[f]$, si dice che il principio di minimizzazione del rischio empirico (ERM) è *consistente* se la differenza tra il rischio atteso calcolato nel minimizzatore del rischio empirico e il rischio atteso ottimale, cioè quello calcolato nel suo minimizzatore, tende a 0 per l che tende ad infinito, cioè se

$$\lim_{l\to\infty}(R[\hat{f}_{H,l}] - R[f_H]) = 0 \tag{1.6}$$

Perciò, nonostante si abbia a disposizione il rischio il empirico, ha senso utilizzarlo solamente se esso è consistente, altrimenti si discosta troppo dal rischio atteso, che è ciò che si vuole minimizzare. È necessario, quindi, andare a investigare quando la (1.6) è verificata studiando tale differenza. Poiché f_H è il minimo per R

$$0 \le R[\hat{f}_{H,l}] - R[f_H] \Leftrightarrow R[\hat{f}_{H,l}] \ge R[f_H] \tag{1.7}$$

Inoltre, dato che $\hat{f}_{H,l}$ minimizza il rischio empirico

$$\begin{aligned} R[\hat{f}_{H,l}] - R[f_H] &= R[\hat{f}_{H,l}] - R_{emp}[\hat{f}_{H,l}] + R_{emp}[\hat{f}_{H,l}] - R[f_H] \\ &\le (R[\hat{f}_{H,l}] - R_{emp}[\hat{f}_{H,l}]) + (R_{emp}[f_H] - R[f_H]) \\ &\le 2\sup_{f\in H}|R_{emp}[f] - R[f]| \end{aligned} \tag{1.8}$$

Dunque, se questo upper bound tende a 0 quando l diverge, per il teorema dei valori intermedi si ottiene la consistenza del principio di minimizzazione del rischio empirico.
Ora, per ottenere un algoritmo operativo in grado di verificare questa condizione, è necessario introdurre il bound di Vapnik-Chervonenkis: con probabilità $1-\eta$, esso stabilisce che il rischio atteso è maggiorato dalla somma tra il rischio empirico e un termine, detto *termine di capacità*, dipendente dal numero di esempi del training set, dalle proprietà dello spazio delle ipotesi e dalla costante η.
In particolare, $\forall f \in H$ e con $0 < A \leq B$z, dove A e B sono costanti dipendenti dalla funzione loss:

$$R[f] \leq R_{emp}[f] + \frac{B-A}{2}\sqrt{\epsilon(l,h,\eta)} \tag{1.9}$$

dove ϵ, il termine di capacità, è definito da

$$\epsilon(l,h,\eta) = \frac{4}{l}\left(h(\ln\frac{2l}{h}+1) + \ln\frac{4}{\eta}\right) \tag{1.10}$$

l è il numero di esempi del training set e h la dimensione di Vapnik-Chervonenkis dello spazio delle ipotesi, un valore che indica la massima dimensione del sottogruppo del training set S che si può predire con esattezza con un predittore $f \in H$.
Poiché ϵ non dipende dalla singola funzione, ma da tutto lo spazio delle ipotesi, è valida anche la seguente disuguaglianza

$$\sup_{f\in H}|R_{emp}[f] - R[f]| \leq (B-A)\sqrt{\frac{1}{l}\left(\ln\frac{2l}{h}+1\right) + \ln\frac{4}{\eta}} \tag{1.11}$$

che ci garantisce, fissato lo spazio delle ipotesi e, di conseguenza, la variabile h, la consistenza del principio di minimizzazione del rischio empirico, dato che il secondo membro della (1.11) tende a 0 quando $l \to +\infty$.
Questa condizione permette anche di evidenziare un'importante osservazione riguardo la complessità del problema. Difatti, aumentando la complessità dello spazio delle ipotesi H, si può sempre ridurre il rischio empirico. Questo perché, nella predizione del training set, l'utilizzo di una funzione lineare, ad esempio, imporrà un forte bias esterno al problema, in quanto prescrive che i dati debbano essere tutti allineati, causando necessariamente degli errori. Ciò può essere contrastato andando ad aumentare il grado dei predittori, in modo tale che la funzione possa adattarsi meglio alla distribuzione degli esempi. D'altro canto, un aumento del grado della funzione e, di conseguenza, del numero dei parametri da cui essa dipende, fa sì che il modello sia poco robusto persino rispetto a piccoli cambiamenti del training set. E ciò comporta un predittore con un'alta varianza e incapacità di generalizzare, fattore fondamentale visto che l'obiettivo non è ottenere il massimo dell'accuratezza sui dati di partenza, ma su quelli ancora da analizzare, che certamente si discosteranno almeno leggermente da quelli usati nell'addestramento. Infatti, considerando la (1.9), aumentare la complessità dello spazio delle ipotesi riduce sì il primo termine, il rischio empirico, ma contemporaneamente fa crescere il secondo termine, il termine di capacità. Questo tradeoff prende il nome di *bias-variance dilemma* ed è centrale nel machine learning. Esso, difatti, dipende unicamente dalla scelta dello spazio delle ipotesi H e, se non viene bilanciato accuratamente, può portare ai due problemi opposti analizzati in precedenza. In particolare, nell'ambito dell'apprendimento automatico, definita *fitting* la capacità di approssimare, i due problemi prendono il nome, rispettivamente, di *underfitting* e *overfitting*. Il primo caso si tratta di una sottostima, ovvero non si riescono a

stimare bene i dati che si hanno. Viceversa, nell'overfitting, cioè quando si sovrastima, ci si sta concentrando fin troppo sulla ricerca di una performance ottimale sul training set, ottenendo così un modello molto instabile che, come visto, si discosta eccessivamente dal minimizzatore del rischio atteso, nonostante possa essere ottimo rispetto al rischio empirico.
Per risolvere il problema, quindi, va cercato un opportuno punto di equilibrio per la complessità dello spazio delle ipotesi, equilibrio che si ottiene adottando la strategia della minimizzazione del rischio strutturale (SRM). Infatti, analizzando la (1.9), il secondo membro prende il nome di *rischio strutturale*, la cui minimizzazione permette di ottenere il più piccolo upper bound possibile per il rischio atteso. In questo caso, però, la ricerca del minimo verrà effettuata proprio sullo spazio delle ipotesi, non dopo che quest'ultimo è stato fissato. Pertanto, operativamente, si procederà prendendo una successione di spazi di ipotesi con dimensione di Vapnik-Chervonenkis crescente, selezionando per ognuno il minimizzatore del rischio empirico. Infine, inserendo nel rischio strutturale i vari minimizzatori appena ottenuti, si cerca il modello in grado di rendere minima la somma tra rischio empirico e termine di capacità, ossia si risolve

$$\min_m [R_{emp}[\hat{f}_{H_m}] + \epsilon(l, h_m, \eta)] \tag{1.12}$$

$$\text{con} \quad H_1 \subseteq \cdots \subseteq H_m \subseteq \ldots \quad \text{tali che} \quad h_i = VC(H_i) < VC(H_j) = h_j \quad \text{dove} \quad i < j$$

Dunque, come si può vedere anche da un punto di vista grafico nella figura 1.2, sommando una funzione che decresce (il rischio empirico) con una che incrementa (il termine di capacità), in funzione dell'aumento di complessità dello spazio delle ipotesi, si può trovare un minimo. Esso corrisponderà ad un certo spazio delle ipotesi con capacità intermedia il cui predittore minimizzante il rischio empirico sarà il modello cercato in grado di evitare sia l'overfitting che l'underfitting.

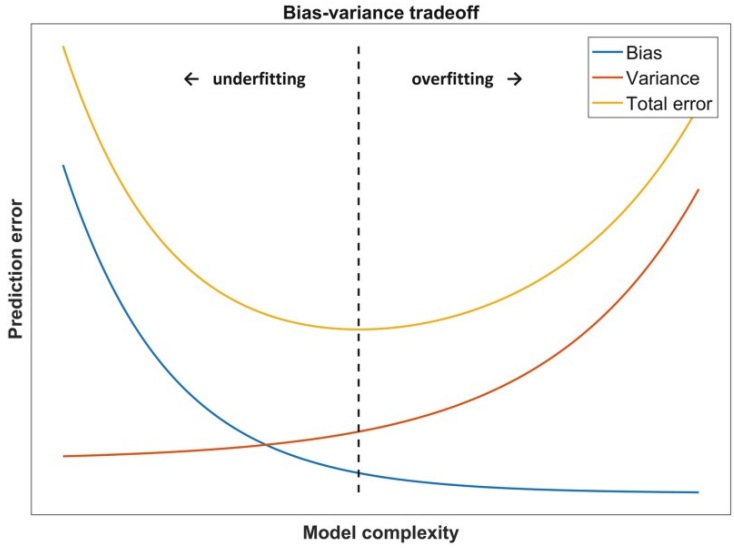


Figura 1.2: Bias-variance dilemma e rischio empirico strutturale

1.3 Reti Neurali

Una particolare branca dell'apprendimento automatico è il *deep learning*, o, in italiano, apprendimento profondo. Essa racchiude al suo interno diverse tecniche di risoluzione di problemi di machine learning che hanno come punto in comune una struttura risolutiva organizzata mediante reti neurali. Le *reti neurali*, già modellizzate a partire dal 1943 ma emerse prepotentemente solo di recente grazie alla maggior potenza computazionale a disposizione, sono chiamate in questo modo perché puntano ad emulare il comportamento del cervello umano nell'acquisizione di informazioni. L'essere umano, infatti, è il più abile a generalizzare dall'esperienza per affrontare situazioni mai viste fino a quel momento, come, ad esempio, completare una frase dove mancano alcune parole. E si è già spiegato quanto questo sia fondamentale per un algoritmo di machine learning. D'altro canto, la macchina è in grado di compiere calcoli e visualizzare dati a una velocità esponenzialmente più alta di quanto l'essere umano sia in grado di fare. Pertanto, riuscire a trovare una struttura capace di sfruttare entrambe queste peculiarità contemporaneamente porterebbe enormi vantaggi in svariate applicazioni, dall'ambito bioinformatico a quello finanziario, da quello meteorologico fino alla guida autonoma, e non solo.
L'architettura della rete neurale si basa sui *layer*: ciò significa che i percettroni, le unità base della rete neurale artificiale che rispecchiano il comportamento dei neuroni, sono organizzati su più livelli. Come è possibile osservare nella figura 1.3, nel percettrone arrivano

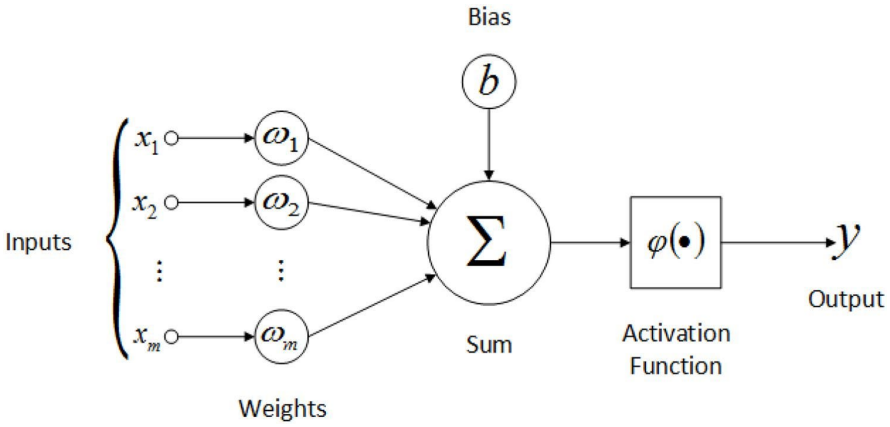


Figura 1.3: Input, output e struttura di un percettrone

n input differenti indicati con x_i (che in base al livello in cui il percettrone si trova possono essere gli input esterni o gli output di altre unità del livello precedente). Ciascuno di essi si lega ad un peso specifico ω_i, detto sinapsi, mediante il prodotto. I risultati di tali moltiplicazioni entrano nella cellula dove vengono sommati, calcolando così il prodotto scalare tra input e pesi. Prima di giungere alla funzione di attivazione, al risultato ottenuto viene aggiunto un bias b specifico per ogni percettrone. La funzione φ ha lo scopo di garantire la non-linearità nel processo, in modo da poter estrapolare dai dati anche quelle relazioni più nascoste altrimenti impossibili da notare. Esistono moltissime funzioni di attivazione differenti a seconda del tipo di output che si vuole ottenere e delle proprietà richieste. Inizialmente, il neurone venne progettato affinché restituisse un output binario. In questo caso la funzione di attivazione era una funzione a soglia, ed il risultato poteva essere solo 1 o 0 a seconda che il prodotto scalare input-pesi fosse, rispettivamente, maggiore o minore del bias. Un neurone così definito veniva chiamato, per l'appunto, unità lineare a

soglia. Successivamente vennero introdotte vere e proprie funzioni di attivazione in grado ottenere predizioni più complesse (nonostante si fosse riusciti a dimostrare che con l'unità lineare a soglia si potesse scrivere qualsiasi formula logica).
Verranno ora presentati più nel dettaglio due specifici esempi di funzione di attivazione in quanto saranno quelle utilizzate in seguito. La prima è la funzione *sigmoide*, descritta dalla funzione $g(t) = \frac{1}{1+e^{-t}}$. Essa ha le proprietà di avere valori compresi tra 0 e 1, potendo così già considerare il risultato una probabilità, di essere monotona crescente e, soprattutto, di essere differenziabile, caratteristica fondamentale per le reti neurali poiché l'apprendimento dall'esperienza è ottenuto mediante algoritmi di ottimizzazione basati sulla discesa stocastica del gradiente. L'altra prende il nome di *ReLU*, che sta per *Rectified Linear Unit*, e ha proprietà completamente differenti. È definita da $g(t) = \max(0, t)$, quindi tutto ciò che è minore di 0 viene ignorato e reso nullo mentre il resto rimane tale. Come si evince dal nome, va sottolineato che questa è una funzione lineare, ma la composizione di vari livelli e varie funzioni di attivazione all'interno della rete neurale garantisce comunque la non-linearità. Infine, oltre a cambiare l'intervallo di valori che può assumere, che qua è $[0, +\infty)$, in questo caso sono monotone sia la g che la sua derivata, che esiste in ogni punto tranne l'origine. La mancanza di differenziabilità in un solo punto viene, in un certo senso, controbilanciata dal punto di vista numerico dal fatto che, a causa dell'aritmetica finita, è molto improbabile che, nella fase di ottimizzazione mediante il gradiente stocastico, una funzione di attivazione venga valutata esattamente in 0. In ogni modo, per ovviare ad alcune particolarità della *ReLU*, sono state ideate anche varianti, come la *Leaky ReLU*, la cui funzione, fissato un parametro a solitamente posto pari a 0.01, o comunque piccolo, è $g(t) = \begin{cases} at & \text{se } x < 0 \\ t & \text{se } x > 0 \end{cases}$. Così facendo vengono mantenuti negativi gli input minori di 0, anche se ridotti grazie al parametro a, si ottiene l'intervallo di output $(-\infty, +\infty)$.

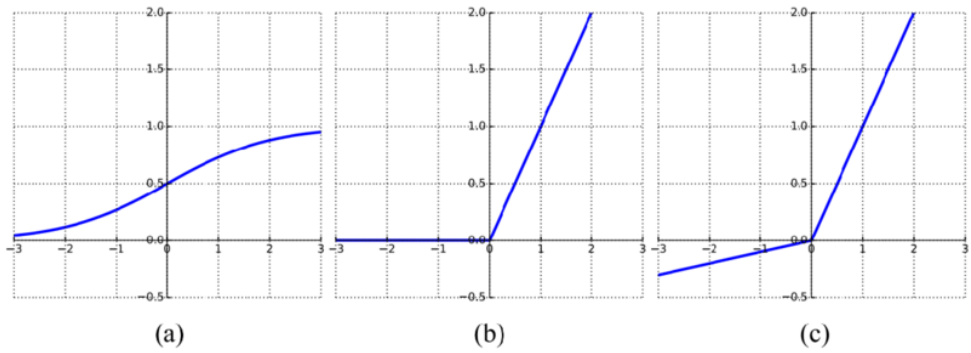


Figura 1.4: Confronto grafico delle funzioni sigmoide (a), ReLU (b) e Leaky ReLU (c)

Come già spiegato nel caso della *ReLU*, per classificare anche insiemi non separabili e più complessi il cui singolo neurone non riesce a gestire, occorre utilizzare una struttura *multilayer*. L'idea è quella di creare una rete neurale a più livelli inserendo, tra il layer di input e quello di output, un numero arbitrario di livelli detti "nascosti", ciascuno dei quali, a loro volta, avrà un numero arbitrario di nodi. Questi valori arbitrari devono essere prestabiliti, così come le funzioni di attivazione e le loss da utilizzare. Essi, infatti, servono a realizzare l'architettura della rete, la quale, una volta costruita, verrà addestra-

ta solamente per settare opportunamente i pesi posti su ciascuna connessione. La scelta di questi valori stabiliti a priori, detti *iperparametri*, è fondamentale per garantire, allo stesso tempo, il giusto bilancio tra la complessità del modello e overfitting. Ma, non essendo eseguita in automatico dalla macchina, diventa complicata e richiede diversi tentativi e tecniche necessarie a testare l'efficacia di svariati modelli, una delle quali è la cross-validation. Quest'ultima può essere implementata in maniere differenti e la più comune è la *cross-validation K-fold*, che consiste nel selezionare una lista di possibili iperparametri e nel suddividere il dataset in K parti di uguale cardinalità. Fatto ciò, ogni K-esima parte del dataset verrà utilizzata come test set e i restanti dati come training set (ed eventualmente validation set), testando su tale frazione di dati ogni combinazione di iperparametri possibili. Infine, facendo la media su ogni fold degli errori commessi da ciascun modello, viene selezionato come predittore ottimale quello che avrà commesso l'errore medio più piccolo. Così facendo, si punta ad ottenere il modello migliore andando ad escludere possibili problemi legati allo sbilanciamento o alle peculiarità del training set in quanto, a turno, ogni dato viene utilizzato K-1 volte per addestrare la rete e 1 è impiegato come test su cui effettuare le predizioni.
Supposto di aver fissato n input, e quindi n nodi iniziali presenti nel primo livello, e di aver costruito una rete con L layer interni oltre a quello di input, si pongono $l = 1, \dots, L$ il layer considerato, x_i l'i-esimo input, $w_{ji}^{(l)}$ il peso sulla sinapsi che va dal nodo i al nodo j del layer l-esimo, $g_j^l : \mathbb{R} \to \mathbb{R}$ la funzione di attivazione del j-esimo nodo al livello l e $w_{j0}^{(l)}$ il bias dello stesso nodo. È possibile così analizzare le operazioni che avvengono all'interno della rete. In particolare, verrà considerato il caso in cui la rete è *fully connected*, così definita in quanto ogni neurone contribuisce con il suo output ai prodotti scalari di tutti i neuroni del livello successivo. Questo è il caso rappresentato nella figura 1.5 ed è la situazione in cui il numero di sinapsi, che corrisponde al numero di pesi, il quale, a sua volta, costituisce il numero di parametri da addestrare, è il più grande possibile. Ma questo, ovviamente, non è l'unico modo per implementare una rete neurale. Infatti, mentre la rete descritta in precedenza viene chiamata *feed-forward*, in quanto le connessioni tra neuroni vanno solo in avanti, da un livello a quello successivo, esistono anche applicazioni in cui è necessario avere memoria di ciò che succede in quanto si lavora su serie temporali. In questo caso si possono creare reti dette *ricorrenti* dove sono presenti connessioni anche tra nodi dello stesso livello o, addirittura, che vanno all'indietro verso nodi precedenti.
Pertanto, in una rete completamente connessa, al primo livello, per ogni neurone j, si hanno il prodotto scalare $a_j^{(1)}$ e l'output $z_j^{(1)}$ così definiti:

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ji}^{(1)} x_i - w_{j0}^{(1)}, \qquad z_j^{(1)} = g_j^{(1)}(a_j^{(1)}) \tag{1.13}$$

mentre dal secondo livello in poi, cioè per $l > 1$, gli input di ogni nodo saranno gli output del livello precedente, e quindi i $z_j^{(1)}$, ottenendo così

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ji}^{(1)} z_j^{(1)} - w_{j0}^{(l)}, \qquad z_j^{(l)} = g_j^{(l)}(a_j^{(l)}) \tag{1.14}$$

dove $N^{(l)}$ indica il numero di neuroni presenti nel livello l.
A questo punto è necessario capire come la rete si organizzi per migliorare i pesi e, cioè,

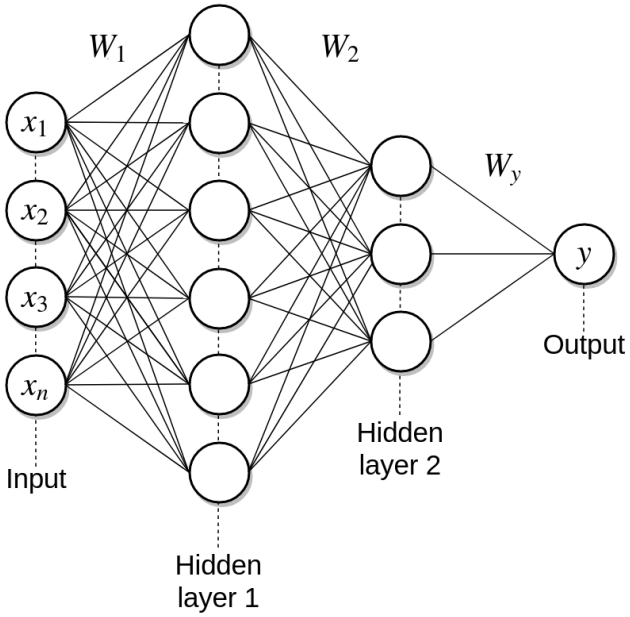


Figura 1.5: Rappresentazione di una rete neurale fully connected con n input e due layer nascosti

quale sia il processo alla base dell'intelligenza della rete. Nel caso delle reti analizzate questo meccanismo prende il nome di *backpropagation*. L'idea, infatti, è quella di utilizzare metodi di ottimizzazione del gradiente stocastico e, calcolato l'errore commesso, questo viene rimandato indietro all'inizio della rete in modo che questa possa aggiornare i pesi, effettuare nuovamente tutti i calcoli, calcolare il nuovo errore e ribilanciare i pesi a seconda di un miglioramento o peggioramento dell'errore commesso, e così via.

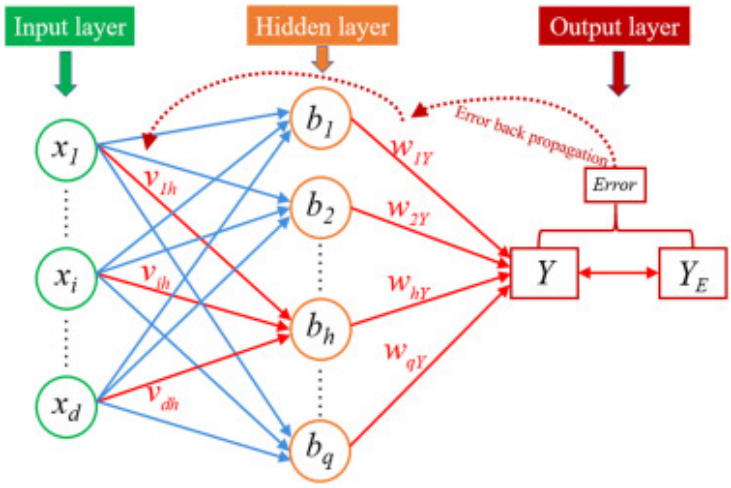


Figura 1.6: Rappresentazione grafica del processo di backpropagation

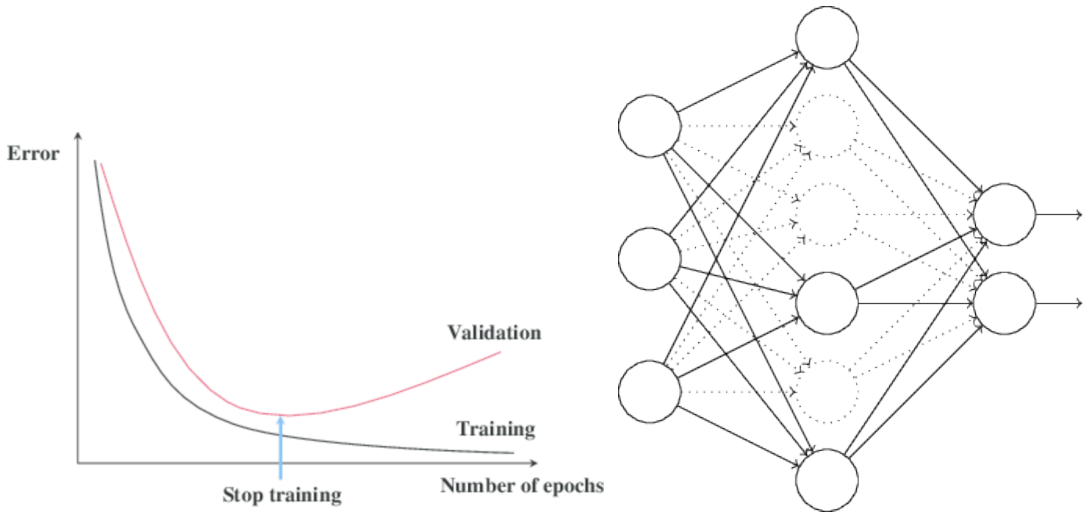
Stabilita una funzione loss $V(x, y, f(x))$ per la misurazione dell'errore della predizione $h(x, \omega)$, dipendente dall'esempio x e dai momentanei pesi ω presenti nella rete, la backpropagation, da un punto di vista matematico, prevede di riscrivere la predizione h in modo tale che dipenda esplicitamente da tutti i pesi della rete. In questo modo, nel momento in cui si va a calcolare il gradiente della loss, si possono aggiornare a ritroso i pesi,

aggiustando prima quelli dell'ultimo livello, che non avranno dipendenze, fino a risalire a quelli del primo livello nascosto, i quali dipenderanno dai valori calcolati nel layer appena aggiornato.
Finora non sono state nominate le features degli esempi del training set, a differenza di quanto fatto con i primi modelli di machine learning. Questo perché il grande vantaggio/-svantaggio delle reti neurali è che, per quanto i dati possano essere forniti all'algoritmo di apprendimento rdotati di features create manualmente dall'uomo, la rete artificiale si comporta come una black box libera di estrarre le informazioni che ritiene più rilevanti senza alcun vincolo. Tale comportamento rappresenta certamente un importante beneficio che il deep learning comporta, ma fa sì, anche, che non si sappia esattamente cosa succeda al suo interno, fatto salvo per gli input e gli output, rendendo complicata la scelta della giusta struttura della rete per ogni specifica mansione. Ciò non significa però che la scelta delle features utilizzate non sia importante: al contrario, costruire dati significativi ed effettuare le opportune operazioni preliminari di pulizia sui dati - questo passaggio prende il nome di *pre-processing* - è cruciale per migliorare sensibilmente le prestazioni di una rete neurale. Un esempio di operazione fondamentale di pre-processing consiste nella standardizzazione del dataset, ottenuta sottraendo ai dati la loro media e dividendo il risultato per la deviazione standard, così da avere dati che possano riflettere una distribuzione normale (la famosa curva a campana di Gauss in grado di rappresentare, infatti, molteplici fenomeni naturali, come l'altezza delle persone del mondo) ed eliminare gli outlier. Questi ultimi sono quei valori anomali che si discostano eccessivamente dalla media, la cui presenza può compromettere l'addestramento della rete facendola soffermare su unico dato che, per quanto possa essere rilevante, è poco comune. Invece è necessario che la rete contempli l'eventuale presenza di eccezioni, sapendo però che l'analisi principale deve esplorare ciò che accade da un punto di vista generico.
Infine, come già accennato, un problema comune delle reti neurali è l'overfitting e diventa impegnativo capire come settare i tanti parametri coinvolti, anche perché essi possono essere molti più del numero di esempi del dataset. Basti pensare che nella rete fully connected che verrà usata in seguito, costituita da livelli con un numero di nodi pari a 256, 512, 1024, 512 e 256, i parametri totali sono 1.358.593, di cui 1.353.473 addestrabili. Per ovviare al problema dell'overfitting esistono diverse soluzioni, di cui tre sono le più utilizzate. La prima è detta *regolarizzazione* e consiste nell'aggiungere alla loss da minimizzare un ulteriore termine, detto per l'appunto *termine di regolarizzazione*, premoltiplicato per una costante λ positiva, la quale costituisce un ulteriore iperparametro da settare mediante cross validation. Un esempio di termine di regolarizzazione è dato dalla funzione di penalizzazione di Tikhonov, che è la norma 2 dei pesi. La funzione da minimizzare, pertanto, diventa

$$\min_{\omega \in \mathbb{R}^m} \sum_{p=1}^{l} V(x_p, y_p, f(x_p)) + \lambda \|\omega\|^2, \qquad \lambda > 0 \tag{1.15}$$

In questo modo la rete non cerca di fare arrivare la loss calcolata sul training a 0, ma cerca di minimizzare questo compromesso dove, più λ è grande, più si cercano di abbassare i pesi a prescindere dal valore della loss.
La seconda soluzione è chiamata *early stopping*. Per capire come essa funzioni è necessario prima chiarire cosa si intenda con epoca. L'algoritmo di addestramento delle rete neurali è un metodo di ottimizzazione iterativo, che ha come funzione obiettivo la loss. Tuttavia, con grandi quantità di dati non si possono processare tutti gli input nello stesso tempo, Al

contrario, una iterata del metodo di ottimizzazione viene applicata alla funzione loss, per esempio quella definita in (1.15), in cui però la sommatoria non viene effettuata su tutti gli esempi del dataset, ma solo su un sottoinsieme, detto *batch*. All'iterata successiva, si definisce un nuovo batch, contenente lo stesso numero di esempi, e si calcola nuovamente l'aggiornamento dei parametri mediante un passo del metodo di ottimizzazione stocastica. Quindi, quello di cui va tenuto conto è quante volte l'intero training set viene osservato dalla macchina: questo fenomeno è ciò che è chiamato epoca. Fissato a priori un numero massimo di epoche che l'algoritmo deve compiere, per evitare che esso si soffermi sul miglioramento delle predizioni sui singoli dettagli del dataset e per velocizzare l'addestramento, viene impostato un valore intero chiamato *patience*: se, dopo un numero di epoche pari al valore patience, la loss calcolata sul validation set non è diminuita, allora si prende come modello quello che aveva dato come valore di loss la più piccola sino a quell'istante. Ovviamente, il numero massimo di epoche e la patience utilizzata sono altri due iperparametri che vanno opportunamente settati.
Un altro modo per prevenire l'overfitting prevede l'utilizzo di un ulteriore iperparametro $p \in (0, 1]$, solitamente posto ad un valore non eccessivamente alto. Questa variabile indica la probabilità di silenziare un neurone e, di conseguenza, di eliminare per quell'epoca certe connessioni. In questo modo non vengono utilizzati sempre tutti i dati, ma di volta in volta alcuni saranno eliminati e poi riattivati. Lo scopo di questo approccio è quello di far sì che il predittore sia stabile ai cambiamenti del training in modo da essere in grado di generalizzare ottenendo buone performance anche su nuovi dati. Questo procedimento prende il nome di *Dropout*.



(a) *early stopping* (b) *dropout*

Figure 1.7a e 1.7b : A sinistra la rappresentazione della tecnica di early stopping, a destra la simulazione di una rete in cui viene applicato il dropout

Un ultimo iperparametro che va analizzato e che è fondamentale nelle reti neurali è il *learning rate*. Anche questo parametro è compreso tra 0 e 1 e ha lo scopo di controllare la velocità di apprendimento della rete. Nel dettaglio, esso indica di quanto scalare l'errore totale calcolato prima di applicare la backpropagation. In altri termini, esso è il parametro di lunghezza di passo che si applica al gradiente della loss, calcolato su un batch

di esempio, per effettuare l'aggiornamento dei parametri nella fase di ottimizzazione mediante il metodo del gradiente stocastico. Esso è un parametro molto delicato, poiché, se troppo elevato, rischia di far oscillare il predittore a causa di passi eccessivamente lunghi tra un'iterazione e l'altra, rendendo così impossibile la ricerca del minimo della funzione obiettivo. Viceversa, un learning rate troppo basso, oltre che ad aumentare il tempo necessario per l'addestramento della rete, rischierebbe di impedire all'algoritmo di migliorarsi a sufficienza facendolo bloccare in un punto non ottimale.

Capitolo 2

Learning to Rank: Significato e analisi degli approcci

In questo secondo capitolo verranno analizzate le fondamenta teoriche alla base del progetto realizzato. Come detto, il lavoro compiuto riguarda un confronto tra possibili implementazioni di metodi risolutivi per un problema di learning to rank. Pertanto, prima di tutto, in questa sezione sarà spiegato cosa si intenda con learning to rank ed i motivi per i quali esso abbia recentemente acquisito popolarità all'interno della comunità scientifica. Esso, infatti, è la base di numerosissime e utilissime applicazioni pratiche, anche le più disparate. In seguito, una volta compreso il quadro del problema da affrontare, lo studio si concentrerà sulle varie opzioni risolutive proposte in letteratura. Di queste verranno illustrati sia gli schemi generali che ne stanno alla base, sia alcune tecniche specifiche, evidenziando le principali differenze ed i punti in comune. In questo modo, quindi, sarà possibile comprendere appieno le scelte approntate nella realizzazione del progetto e la relativa discussione in merito ai risultati ottenuti.

2.1 Il ranking nell'Information Retrieval

Con il termine *Information Retrieval*, che può essere tradotto in italiano con *reperimento dell'informazione*, si intende un insieme di tecniche matematico-informatiche volte a recuperare, gestire ed, infine, restituire una serie di oggetti contenenti precise informazioni. Lo scopo di queste tecniche, quindi, consiste nel soddisfare le richieste di conoscenza di un utente in merito ad un certo argomento. Pertanto, le due componenti principali di un sistema di information retrieval sono le query e gli oggetti. Con il termine *query* si indicano le domande che vengono poste al sistema da parte dell'utente. Quest'ultimo, mediante il sistema che funge da intermediario (come può essere un motore di ricerca tipo Google), tramuterà la sua richiesta in una serie di stringhe di parole chiave. In risposta, il meccanismo di information retrieval restituirà tutti quegli oggetti che ritiene possano soddisfare la domanda ricevuta. La natura di questi *oggetti*, che sono la seconda componente principale del processo, spazia tra varie possibilità: documenti di testo, pagine web, oggetti multimediali, email, libri e molto altro. L'importante è che, secondo il sistema, essi siano in grado di soddisfare la richiesta dell'utente. Il processo che è stato descritto prende il nome di *task*.
L'information retrieval è coinvolta in moltissime altre discipline, sia scientifiche, come

l'informatica e la psicologia cognitiva, sia umanistiche, come la linguistica e la semiotica, ed è utilizzata nella vita di tutti i giorni, dal momento che nelle biblioteche pubbliche e nelle università vengono utilizzati sistemi basati proprio su tali tecniche per reperire i libri e le pubblicazioni desiderate. Tornando all'esempio del motore di ricerca, però, è chiaro che un sistema costruito per realizzare task, se implementato esclusivamente nella maniera in cui è stato appena descritto, diventa inutilizzabile. Basti pensare, infatti, che a marzo 2023 il numero di pagine web stimate nell'intera Rete supera i 50 miliardi [16]. Evidentemente, se in merito ad una precisa richiesta vengono restituiti tutti i documenti che possono risultare affini senza compiere una selezione, accontentandosi di fornirli in un qualunque ordine, diventa impossibile per l'utente capire quale siano i documenti più importanti da considerare nella ricerca. È qui allora che interviene il ranking.
Il *ranking* è, infatti, il problema centrale dell'information retrieval. Esso significa "ordinamento" e indica la necessità di fornire i documenti, individuati inizialmente in maniera casuale durante la raccolta, secondo un criterio di classificazione che disponga ai primi posti gli oggetti più rilevanti e, via via, quelli meno significativi. Questo criterio deve essere basato sul loro grado di affinità con la query da soddisfare. Questo tipo di problema è salito sempre più prepotentemente alla ribalta negli ultimi anni, in quanto il numero di dati che si ha a disposizione, di per sè enorme, aumenta sempre più velocemente, così come altrettanto fa l'utenza in grado sia di creare dati che di richiedere query. Tutto ciò, ha portato a dover perfezionare questi strumenti, non solo approfondendo lo studio delle tecniche già esistenti, ma anche proponendo altre strade, come l'utilizzo del machine learning. Quest'ultimo, per poter essere addestrato e fornire performance accettabili, necessita proprio di quell'enorme quantitativo di dati di cui ora si è a disposizione. Sono svariate, infatti, le applicazioni quotidiane che fanno uso del ranking nell'information retrieval: oltre ai già citati motori di ricerca (il sito *Internetlivestats* stima che Google elabori 3,5 miliardi di ricerche ogni giorno per un totale di ben 40.000 domande al secondo), anche i sistemi di raccomandazione, presenti ad esempio nella piattaforma di Amazon, utilizzano il profilo e la cronologia degli utenti per suggerire gli articoli più idonei all'acquisto per ciascuno, così come i siti delle agenzie di viaggio, tipo Booking, filtrano i migliori alberghi da suggerire, a seconda dei dati personali e delle condizioni inserite relative a date o altre preferenze.
Come è già stato accennato, esistono due possibili approcci principali per risolvere il problema del ranking. I primi sono più semplici in quanto prevedono il calcolo di certe quantità precise, utili per poter poi effettuare un determinato tipo di ordinamento. I secondi, invece, fanno uso del machine learning e possono essere considerati un'evoluzione dei precedenti per due motivi. Innanzitutto, perché le prime tecniche sono parametriche e prevedono la necessità di un ulteriore studio riguardo al settaggio di tali parametri, cosa che invece il machine learning riesce a compiere in maniera automatica e più efficientemente, come spiegato nel primo capitolo. In secondo luogo, le tecniche di machine learning possono memorizzare i valori delle quantità calcolate in precedenza per combinarli utilizzandoli come features dei documenti da classificare, creando così modelli più sofisticati. Il learning to rank, allora, racchiude tutti quegli approcci che sfruttano il machine learning per risolvere il problema del ranking nell'information retrieval. Pertanto, sia per mostrare l'evoluzione storica di queste tecniche, di cui il learning to rank rappresenta lo stadio più recente, sia, soprattutto, perché questa disciplina - il learning to rank, appunto - utilizza come features anche i valori calcolati dalle precedenti tecniche, ha senso approfondire brevemente i ragionamenti che stanno alla base di questi primi metodi.

In ogni caso, l'idea generale che sta alla base di un qualsiasi modello di ranking prevede l'implementazione di una certa funzione f in grado di calcolare un punteggio di rilevanza. Così facendo, per ogni input $x = (q, d)$, che rappresenta la coppia query e documento ad essa associato, si ottiene un punteggio di rilevanza s dato da $s = f(x)$. Quindi, calcolati i punteggi per ogni input del dataset, per ogni query presa in esame verrà stilata una classifica di rilevanza dei documenti. Essa sarà ottenuta ordinando questi ultimi in maniera discendente rispetto al punteggio ad essi attribuito, così da avere ai primi posti gli oggetti di maggior interesse.
Tornando all'analisi più dettagliata delle tecniche che non fanno uso del machine learning per risolvere il problema del ranking, esse possono essere suddivise in due sotto-categorie, a seconda della loro dipendenza o meno dalla query specifica.

2.1.1 Modelli dipendenti dalla query

I primissimi modelli ideati per il ripescaggio dei documenti non soddisfacevano esattamente le richieste del ranking, in quanto essi erano in grado solamente di affermare se un documento fosse significativo o meno, senza attribuirvi un grado di rilevanza. Un esempio di questo tipo è dato dal *modello Booleano*, che sfrutta il numero delle occorrenze dei termini presenti nella query all'interno del documento per stabilire se esso possa essere considerato rilevante.
Un'idea più avanzata consiste, invece, nell'approfondimento di come si possano rappresentare nello spazio query e documenti. In questo modo sarebbe più facile capire quale oggetto ha più similarità con la query. In particolare, il metodo VSM, acronimo di *Vector Space Model*, dopo aver trasformato nella maniera che più si ritiene opportuna i dati in vettori, suggerisce di utilizzare lo spazio Euclideo per rappresentarli. Questo perché, semplicemente calcolando il prodotto scalare tra query e documento, è possibile ottenere un valore di similarità: questa grandezza prende il nome di *cosine similarity*. Infatti, a partire dal prodotto scalare, definito da $\mathbf{A}\cdot\mathbf{B} = ||\mathbf{A}||\ ||\mathbf{B}|| \cos\Theta$, dove $\mathbf{A}$ e $\mathbf{B}$ sono i vettori, è sufficiente dividere per le loro lunghezze per ottenere il coseno dell'angolo θ tra essi compreso. Quindi, la formula della cosine similarity è data da

$$S_C(\mathbf{A}, \mathbf{B}) := \cos\Theta = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.1}$$

Pertanto, un valore di S_C pari a 1 indica che i due vettori sono coincidenti, così come se il risultato fosse -1, ciò significherebbe che i due vettori sarebbero opposti. Ricordando che il coseno pari a 0 significa che i due vettori sono ortogonali, è poi possibile capire l'interpretazione dei valori intermedi. Il VSM, quindi, permette di ottenere un indice di similarità indipendente dalla grandezza dei vettori e che richiede un basso costo computazionale. Per quanto riguarda la rappresentazione vettoriale richiesta dal VSM esistono vari modi per ottenerla e la più utilizzata è la funzione peso TF-IDF. Dall'inglese *Term Frequency–Inverse Document Frequency*, lo scopo della TF-IDF è calcolare l'importanza di un termine all'interno del documento. Questo valore è costituito da due componenti, la TF e la IDF, appunto. La prima misura le occorrenze del termine studiato t all'interno del documento d, normalizzando poi per il numero di parole che il documento stesso contiene, così da non avvantaggiare quelli più lunghi. Pertanto

$$\mathrm{TF}(t, d) = \frac{n_{t,d}}{|d|} \tag{2.2}$$

dove $n_{t,d}$ indica il numero di apparizioni del termine t nel documento d e $|d|$ la dimensione di d espressa come numero di parole. Il secondo fattore, la IDF, cresce in maniera inversamente proporzionale al numero di documenti che contengono quel termine t ed è descritta dalla relazione

$$\mathrm{IDF}(t) = \log \frac{N}{n(t)} \tag{2.3}$$

in cui N equivale al totale dei documenti presenti nella collezione e $n(t)$ corrisponde al numero di documenti che contengono il termine t. In questo modo, calcolando la TF-IDF come

$$\text{TF-IDF}(t, d) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t) \tag{2.4}$$

si ha una funzione che premia la coppia termine-documento quando il primo è ripetuto molte volte nel secondo, ma, al tempo stesso, penalizza le situazioni in cui quel termine è presente in svariati documenti. Questo perché ha senso che un documento venga considerato rilevante per la query se contiene le parole per cui la query deve indagare, ma solo se tali termini sono specifici per la task da soddisfare. Se nella query, ad esempio, sono contenute delle preposizioni semplici o delle congiunzioni, queste saranno certamente ripetute molte volte in un documento, ma proprio perché fanno parte della struttura sintattica di una frase e sono contenute in pressoché ogni testo, esse non daranno informazioni riguardo alla query specifica.
Un altro spazio utilizzato è lo *spazio semantico latente*. In questo caso si fa uso della tecnica della decomposizione a valori singolari per originare uno spazio in cui non valga più l'assunzione che la VSM implicava per come era progettata, ossia il fatto che tutti i termini compaiano con una probabilità indipendente l'uno rispetto all'altro. E questa assunzione è logicamente falsa, sia a causa delle regole grammaticali presenti in una lingua (ad esempio, nella lingua inglese dopo "to" è probabile che compaia un verbo all'infinito), sia per il significato di ogni parola (è improbabile trovare la parola "festa" in un paper scientifico che riguarda la medicina). Anche in questo caso, poi, la similarità viene calcolata attraverso l'utilizzo del coseno.
Un altro approccio riguardante le tecniche dipendenti dalle query prevede l'adozione di un framework probabilistico. Tra questi, il modello più diffuso è chiamato *BM25*, che sta per *Best Matching*. In realtà, essendo parametrico, esso racchiude al suo interno una famiglia di modelli, a seconda di come lo si implementa. In particolare, data la query q, che contiene i termini $t_1, \ldots, t_M$, l'implementazione più comune del punteggio BM25 per il documento d è data da

$$\mathrm{BM25}(d, q) = \sum_{i=1}^{M} \frac{\mathrm{IDF}(t_i) \cdot \mathrm{TF}(t_i, d) \cdot (k_1 + 1)}{\mathrm{TF}(t_i, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\mathrm{avdl}}\right)} \tag{2.5}$$

dove $\mathrm{TF}(t_i,\ d)$ è la frequenza dell'i-esimo termine t_i nel documento d come definita nella (2.2), $|d|$ è la dimensione di d espressa come numero di parole, avdl è la lunghezza media dei documenti della collezione, k_1 e b sono parametri liberi che vanno settati e $\mathrm{IDF}(t_i)$ è la (2.3). Lo scopo di BM25, quindi, è il calcolo di un punteggio di significatività di un documento ottenuto tramite il logaritmo della probabilità di rilevanza del documento stesso.

2.1.2 Modelli non dipendenti dalla query

A differenza dei metodi precedenti, questi modelli cercano di ordinare i documenti indagando esclusivamente la loro importanza, a prescindere dalla query considerata. Uno degli esempi più famosi in questo senso è dato da *PageRank*. Brevettato nel 2001 dalla Stanford University, esso è stato per lungo tempo il cardine delle operazioni di ranking impiegate da Google per restituire i risultati di una ricerca e tuttora fa parte del complesso e moderno algoritmo utilizzato dal celebre motore di ricerca. Questo modello, infatti, è perfetto per essere implementato per il ranking di pagine web poiché utilizza i collegamenti ipertestuali, cioè i link che collegano le varie pagine, per determinare l'autorevolezza di ciascuna di esse. In sostanza, l'idea che sta alla base di PageRank è che se un documento è citato da molti altri, cioè molti di questi hanno link che puntano alla pagina considerata, allora esso sarà tanto più rilevante quante più sono le pagine ad esso subordinate. Non solo: anche l'importanza di chi ti cita è importante. Infatti, il link proveniente da una pagina rilevante avrà molto più valore di un collegamento che parte da un documento poco famoso e non autorevole, soprattutto se i link in uscita dalla pagina con un buon livello di considerazione sono pochi, perché ciò significa che gran parte delle informazioni che contiene derivano proprio dalla pagina a cui punta.
Da un punto di vista implementativo, PageRank si propone di calcolare la probabilità di arrivare ad un certa pagina d_f cliccando casualmente su un link di un'altra pagina d_i, motivo per cui il calcolo viene effettuato sommando per tutte le restanti pagine della collezione B l'inverso della quantità di link che ciascuna contiene, indicata con $U(d_i)$. Come detto, però, l'autorevolezza di un documento "passa" dalla pagina di partenza a quella di arrivo, motivo per il quale il numeratore si moltiplica per il valore di PageRank del documento d_i. In realtà, nella formula finale di PageRank è presente anche un secondo fattore ed un parametro α. Questo parametro è detto *damping factor*, è solitamente settato a 0.85 ed indica la probabilità che si continui la ricerca passando da un link all'altro. In questo modo, (1-α), rappresenta la piccola possibilità che la ricerca venga effettuata entrando casualmente in una pagina, senza passare da un link esistente. Questa è la situazione in cui si sceglie una prima pagina web dopo aver effettuato una query o si apre casualmente un indirizzo web. Pertanto, la probabilità di finire su un preciso documento mediante questa strada si otterrà dividendo il suddetto termine per il numero totale N di documenti nel web. La formula finale sarà quindi

$$\mathrm{PR}(d_f) = \alpha \sum_{d_i \in B} \frac{\mathrm{PR}(d_i)}{U(d_i)} + \frac{(1-\alpha)}{N} \tag{2.6}$$

Va sottolineato che l'algoritmo di PageRank è iterativo: ciò significa che i documenti hanno un punteggio PR iniziale pari al damping factor e, passo dopo passo, il valore verrà aggiustato fluendo da un documento all'altro. Un esempio di raffigurazione finale del processo si trova nella figura 2.1.

2.2 Le metriche di valutazione per il ranking

Dal momento che è già stato descritto un considerevole numero di algoritmi di ordinamento e ricordando che, rispetto alle più recenti strategie di learning to rank che verranno analizzate nel dettaglio nella sezione seguente, essi rappresentano solo una piccola percentuale dei possibili metodi risolutivi del problema considerato, sorge spontaneo chiedersi

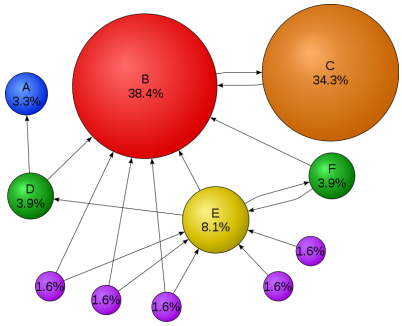


Figura 2.1: Esempio di valori finali di PageRank all'interno di una collezione di documenti al termine dell'algoritmo

in che modo si possa decidere quale algoritmo sia migliore rispetto ad un altro. La risposta è fornita dalle *metriche di valutazione*, la cui descrizione sarà l'argomento di questo sottocapitolo.
Innanzitutto, è necessario capire come funzioni una metrica di valutazione in generale. Lo scopo, infatti, è la verifica dell'accuratezza delle previsioni che l'algoritmo realizza sul test set rispetto all'ordinamento originale. Una peculiarità di tutte le metriche di valutazione che verranno presentate risiede nel fatto che esse sono formulate per essere utilizzate su una singola query. Perciò, dopo aver collezionato un insieme di query ed i relativi documenti (non necessariamente lo stesso numero di documenti per ciascuna interrogazione), si seleziona una certa richiesta q. A questo punto si raccolgono tutti i documenti $\{d_j\}_{j=1}^{m}$ legati alla query considerata, tenendo in memoria il loro punteggio di rilevanza fornito dall'esterno. Esistono diverse possibilità per ottenere il ranking reale, come un intervento a priori dell'uomo o, nel caso di motori di ricerca, il numero di accessi effettuati ad una determinata pagina a seguito di una particolare richiesta, premiando come più rilevanti quelle che ne hanno ricevuti di più. Successivamente, interviene l'algoritmo vero e proprio, il quale predice la significatività di ogni documento e propone un certo ordinamento. Quest'ultimo verrà allora confrontato con quello veritiero mediante la metrica di valutazione, che fornirà un certo punteggio all'algoritmo. Infine, ripetendo il procedimento appena spiegato per ogni query, sarà sufficiente effettuare una media tra tutti i punteggi ottenuti per poter assegnare al modello uno score generale.
È quindi fondamentale che l'attribuzione del punteggio di rilevanza, effettuato a priori ed assunto come veritiero, sia effettivamente accurata. Quindi, visto che affidare all'essere umano questo compito può comportare un eccessivo aumento del tempo impiegato, agli approcci automatizzati di assegnazione della rilevanza si richiede che due proprietà in particolare vengano soddisfatte: la completezza e la consistenza. La *completezza*, infatti, misura quanti documenti rilevanti sono stati effettivamente raccolti ed inseriti nella collezione. La *consistenza*, invece, si occupa di verificare se tutti i documenti veramente rilevanti siano stati contrassegnati come tali e, analogamente, se ciò si è verificato anche per quelli irrilevanti.
Prima di andare ad investigare l'implementazione effettiva delle metriche di valutazione bisogna comprendere come possano essere attribuiti ad uno o più documenti questi punteggi di rilevanza. In particolare, in letteratura esistono tre approcci principali, i quali si differenziano in base a quanti documenti vengono considerati contemporaneamente per assegnare le relevances.

- **Punteggio singolo**. Ad ogni documento viene associata una variabile Booleana (cioè un variabile binaria, del tipo 1 o 0, oppure *True* o *False*) che indichi se esso è rilevante oppure no. Esistono anche sistemi più complessi in grado di attribuire persino un grado di rilevanza al singolo documento assegnandogli, ad esempio, un intero compreso tra 0 e 4, estremi inclusi. In questo caso i numeri corrispondono ad un punteggio di rilevanza crescente che corrisponde, ad esempio, ai voti *Pessimo*, *Discreto*, *Buono*, *Ottimo* e *Perfetto*. Pertanto, assegnato ad ogni documento d_j relativo ad una query q un punteggio di rilevanza l_j, si dirà che il documento d_u è più rilevante di d_v se $l_u > l_v$, così da risalire alla classifica globale semplicemente considerando tali relazioni.

- **Punteggio binario**. Questa strategia si occupa di indagare i documenti a coppie, generando in risposta come valori solo 0 e 1, a seconda di quale dei due input considerati sia il più rilevante. In particolare, se il documento d_u è più rilevante di d_v, la label $l_{u,v}$ sarà pari a 1, altrimenti si otterrà $l_{u,v} = 0$. In questa implementazione è meno immediato risalire all'ordinamento globale all'interno di una query. Tuttavia è comunque possibile farlo valutando per ogni documento quanti "scontri" vincerebbe considerando ogni possibile accoppiamento.

- **Punteggio globale**. In quest'ultima situazione non ci si preoccupa di assegnare un punteggio specifico ad ogni documento, ma viene solamente fornito il ranking globale o parziale relativo a ciascuna query. Per ogni gruppo di documenti $\{d_j\}_{j=1}^{m}$ relativi ad una query q, questo valore viene solitamente restituito come una permutazione π_l o, eventualmente, come un insieme di permutazioni.

L'approccio proposto dall'ultima strategia, nonostante non si preoccupi che il punteggio assegnato ad ogni documento sia esatto, è comunque corretto. Bisogna ricordare, infatti, che l'obiettivo del ranking è l'ottenimento della corretta classificazione dei documenti, indipendentemente da come essa venga ottenuta o dai singoli punteggi. Difatti, basti pensare ad un semplice esempio: si hanno a disposizione tre documenti, A, B e C i cui punteggi di rilevanza reali sono, rispettivamente, 3, 4 e 1. Se l'algoritmo utilizzato, prima di realizzare l'ordinamento, attribuisse ai tre esempi i punteggi 2, 3 e 0, il ranking dato in output sarebbe comunque corretto, nonostante tutti i punteggi fossero sbagliati. Ciò che conta nella valutazione di un algoritmo di ranking, pertanto, è esclusivamente l'ordine dei documenti dato in output. È questo, quindi, ciò che una metrica di valutazione per il ranking deve tenere in considerazione. Saranno ora presentate dettagliatamente proprio alcune di queste metriche.

Mean Reciprocal Rank (MRR): Questa è la più semplice di tutte le metriche e si occupa di controllare esclusivamente in che posizione viene classificato il primo documento realmente rilevante, tralasciando così come vengono ordinati i restanti dati significativi. Detta $r(q)$ la posizione del primo documento significativo, si ha che

$$\mathrm{MRR} = \frac{1}{r(q)} \tag{2.7}$$

Quindi, se esso si trova al primo posto la MRR varrà uno, altrimenti avrà un valore più piccolo sempre più vicino a 0 tanto più questo documento sarà classificato lontano dal primo posto.

Mean Average Precision (MAP): La MAP si utilizza in situazioni in cui le label dei documenti sono date come punteggi singoli Booleani. Essa combina due grandezze differenti, ossia la precisione (P) e la sensibilità (R). La *precisione* (in inglese, precision) è il rapporto tra il numero di documenti rilevanti ed il totale considerato, quindi un modello con un'alta precisione sbaglia poche volte le predizioni. D'altro canto, esso potrebbe non essere sensibile, il che significa che potrebbe dare luogo a molti falsi negativi. Essi si manifestano quando un documento corretto viene classificato come irrilevante. Difatti, non rientrando nella collezione considerata dalla precisione, questo tipo di errore non inficia la precisione. In questo caso, allora, interviene la *sensibilità* (in inglese, recall). Essa è data dal numero di previsioni corrette sul totale di tutti i documenti presenti effettivamente. Al contrario della precisione, un'alta sensibilità potrebbe comunque causare un elevato numero di falsi positivi, ossia di documenti irrilevanti individuati come significativi. Ciò evidenzia il motivo per cui queste due grandezze vadano combinate insieme per ottenere un modello efficiente.
In particolare, nel caso della MAP, vengono calcolate P_k e R_k, dove il pedice k indica che esse sono limitate alle prime k posizioni. Quindi, per $k = 1, \ldots, n$ e per la query q, si ottiene che la formula per l'average precision, detta AP, è data da

$$\mathrm{AP}(q) = \sum_{k=1}^{n} (R_k - R_{k-1}) P_k \tag{2.8}$$

dove R_0 vale 0. Questa quantità combacia con l'area sottesa alla curva *presion-recall*, come si vede dalla figura 2.2 e, matematicamente, scorrendo tutta la classifica, calcola le percentuali di documenti rilevanti inseriti nelle prime k posizioni e tali valori li somma solo quando un nuovo documento significativo compare in classifica, per poi pesare tutto rispetto al numero di documenti totali. Infine, poiché la AP dipende dalla specifica query, facendone la media su tutte le interrogazioni q si ottiene la MAP.

Figura 2.2: Rappresentazione della curva precision-recall e di un esempio di calcolo della MAP

Discounted Cumulative Gain (DCG): A differenza delle metriche precedenti, la DCG è dotata di una proprietà fondamentale nell'ambito del ranking. Innanzitutto, può essere utilizzata anche quando i punteggi singoli rappresentano un indice di rilevanza, cioè quando la classi a cui assegnare un documento non sono esclusivamente rilevante-irrilevante ma descrivono un grado di significatività. Soprattutto, questa metrica contiene al suo interno un esplicito fattore di penalizzazione dipendente dalla posizione a cui

appartiene il documento predetto in maniera errata. Per capire l'importanza di questa caratteristica, è possibile prendere come esempio, ancora una volta, il caso del motore di ricerca di Google: quando un utente effettua una ricerca, la stragrande maggioranza delle volte considera solo i primissimi documenti che vengono riportati, quasi mai andrà a visualizzare la seconda pagina dei risultati offerti da Google, tanto meno le schede successive. Questo permette di capire la necessità di posizionare nelle prime posizioni i documenti più rilevanti e, soprattutto, quanto conti la posizione in cui viene commesso l'errore. Difatti, se, per esempio, nella quinta di pagina di Google due documenti sono invertiti, ciò non importerà praticamente a nessuno. Ma, al contrario, un errore nelle prime posizioni inficerà la ricerca di qualsiasi utente. È essenziale, quindi, penalizzare maggiormente queste ultime tipologie di errori. Queste due necessità saranno rappresentate dai due fattori che costituiscono la DCG, ossia la funzione esponenziale G_k ed il fattore di sconto logaritmico D_k.
Nello specifico, data una lista $D = \{d_1, \ldots, d_n\}$ di n documenti relativi ad una query q, proprio perché i documenti su cui va posta maggiore attenzione sono quelli predetti nelle prime posizioni, la DCG si concentra solo sull'esattezza dell'ordinamento dei primi k documenti (infatti viene indicata con DCG@k (q)) ed è definita come segue

$$\mathrm{DCG@}k\,(q) = \sum_{j=1}^{k} \mathrm{G}_j \mathrm{D}_j \tag{2.9}$$

dove i due fattori sono dati da

$$\mathrm{G}_k = 2^{y_k} - 1 \qquad\qquad \mathrm{D}_k = \frac{1}{\log_2(k+1)} \tag{2.10}$$

Il primo, quello esponenziale, è il *fattore di guadagno*. Esso cresce all'aumentare della rilevanza effettiva del documento, ossia la sua etichetta, che infatti è rappresentata da y_k. In questo modo, maggiore è l'importanza del documento più elevato sarà il suo punteggio, perché essendo più rilevante si richiede che esso si collochi nelle primissime posizioni. Viceversa, minore sarà la relevance del documento, inferiore sarà la sua importanza e, di conseguenza, il suo punteggio, tant'è che nella formula la sottrazione dell'unità serve proprio per attribuire un valore pari a 0 a tutti quei documenti che hanno 0 come etichetta. Il secondo elemento, invece, è il *fattore di sconto*. Il suo obiettivo è diminuire il peso dei documenti man mano che la loro posizione in classifica aumenta, così che gli errori commessi nelle prime posizioni siano più incisivi. Infatti, per come è definito, D_k varrà 1 alla prima posizione e poi, man mano, calerà con andamento logaritmico, così che all'inizio la discesa sia più rapida, mentre alla fine, dal momento che già ci si trova nelle retrovie della classifica, il cambio di una posizione non sia determinante, come invece accade nelle prime posizioni. In definitiva, poiché poi i due fattori sono da moltiplicare, maggiore sarà il valore della metrica migliore sarà stato il comportamento del predittore.
In realtà, per garantire che il punteggio finale sia compreso tra 0 e 1, si utilizza frequentemente la variante detta NDCG, che sta per *Normalized Discounted Cumulative Gain*. Per ottenere il valore di quest'ultima metrica, è necessario pre-calcolare il punteggio della DCG della predizione ideale, ossia quello ottenuto nel caso in cui i documenti siano classificati alla perfezione, in base alle loro reali rilevanze. Questo, pertanto, sarà il massimo valore ottenibile dalla DCG per quella lista di documenti. Quindi, per ottenere la NDCG, è sufficiente calcolare la DCG della previsione come spiegato in precedenza e, successivamente, dividere tale valore per il punteggio ideale.

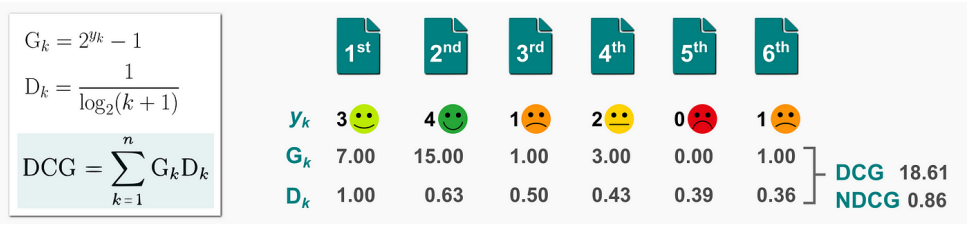


Figura 2.3: Esempio di calcolo della DCG

Ancora una volta, poiché come la MAP la DCG e la NDCG sono delle metriche che vengono calcolate sulle singole query, per ottenere un indice di valutazione della performance globale del modello, si devono mediare i punteggi delle metriche rispetto a tutte le query del dataset. Questo garantisce che, nelle suddette metriche, ogni interrogazione abbia lo stesso peso delle altre, così che non vi siano query che sovrastano le altre. In questo modo, anche se i documenti di una certa query q sono stati etichettati non correttamente, ciò non inficerà più di tanto il risultato globale restituito dalla metrica.
Esiste però un aspetto negativo legato alle metriche che, come la DCG, fanno un uso esplicito della posizione dei documenti. Infatti, dal momento che piccoli cambiamenti nei punteggi predetti dal modello non comportano necessariamente uno scambio di documenti, soprattutto nel caso della regressione, questo tipo di metriche, solitamente, sono non continue e non differenziabili rispetto a tali score. Ma, dato che spesso si utilizzano metodi di ottimizzazione legati alla discesa del gradiente, questo fatto può causare diversi problemi, i quali verranno trattati in seguito, di volta in volta, nei singoli algoritmi.

Spearman correlation: Un'ultima metrica, che ha un supporto teorico diverso dalle precedenti in quanto deriva dalla statistica, è la *Spearman correlation*. Il coefficiente di correlazione di Spearman è una misura statistica di correlazione non parametrica che viene utilizzata per valutare la relazione tra due variabili ordinali. Esso è ottenuto come caso particolare del coefficiente di correlazione di Pearson, il quale, date due variabili statistiche, è definito come la loro covarianza divisa per il prodotto delle deviazioni standard delle due variabili.
Come già osservato, nei problemi di ranking, avendo moltissimi documenti e, solitamente, 5 gradi di rilevanza, si verificano diverse situazioni di "pareggio", nel senso che i documenti possono avere posizioni interscambiabili in quanto dotati della stessa etichetta. Quindi, dato che in presenza di valori non univoci la formula utilizzata per calcolare la Spearman correlation si complica parecchio, essa non viene riportata. Se necessario, infatti, esistono librerie apposite che permettono di calcolarla gestendo questo tipo di situazioni.
Da un punto di vista interpretativo, il coefficiente di correlazione di Spearman può assumere un valore compreso tra -1 e 1, dove un valore pari a 1 indica una perfetta correlazione positiva tra le due classifiche (cioè le classifiche sono identiche), lo 0 indica una mancanza di correlazione e un valore di -1 indica una perfetta correlazione negativa (cioè le classifiche sono invertite). In generale, un coefficiente di correlazione di Spearman elevato indica una forte similarità tra le due classifiche, mentre un coefficiente basso indica una bassa similarità o, addirittura, una differenza significativa tra le due classifiche. Pertanto, nei problemi di ranking, il coefficiente di correlazione di Spearman è un'utile metrica in grado di valutare la validità e l'affidabilità delle classifiche ottenute da diversi metodi o fonti. Difatti, in queste situazioni, questo coefficiente permette di confrontare la similarità tra

l'ordinamento predetto e quello reale.
Il motivo per cui viene preferito Spearman a Pearson risiede nel fatto che il primo è "non parametrico": ciò significa che esso è in grado di determinare una perfetta correlazione tra due variabili X e Y ogniqualvolta tra le due è presente una qualsiasi relazione monotona. Al contrario, il coefficiente di Pearson è pari a 1 solo quando la relazione tra i due campioni è lineare, quindi l'utilizzo di Spearman è preferito perché è in grado di generalizzare maggiormente.
Anche per quest'ultima metrica va sottolineato che i ragionamenti presentati sono validi per le singole query. Ciò significa che, ancora una volta, per calcolare la prestazione globale del modello è necessario considerare la media dei punteggi relativi alle singole interrogazioni.

2.3 Learning to Rank

Come spiegato in precedenza, il machine learning è diventato sempre più centrale con il passare del tempo nei problemi di classificazione. Innanzitutto perché esso permette di regolare in maniera più rapida e più efficiente tutti quei parametri da cui dipendono i metodi di ranking spiegati in precedenza, come le variabili k_1 e b utilizzate nella (2.5) per calcolare la BM25 o lo scalare α della (2.6) necessario per valutare i valori di PageRank dei documenti. Questa messa a punto dei parametri, a causa della non continuità e della non differenziabilità dei modelli rispetto a tali valori, può risultare tutt'altro che banale. Inoltre, un secondo aspetto cruciale a cui il machine learning riesce a dare risposta riguarda la capacità di combinare tutti questi metodi. In questo modo, l'algoritmo creato dal machine learning è in grado di sfruttare i punti di forza dei modelli precedenti originandone uno nuovo ancora più efficiente e capace di migliorarsi continuamente grazie agli innumerevoli feedback che riceve, considerando per esempio il caso dei motori di ricerca, dalle visite giornaliere degli utenti alle varie pagine web.
Quindi, con il termine *learning to rank*, si indica l'insieme di tutti quegli algoritmi che cercano di risolvere il problema del ranking mediante l'utilizzo del machine learning. In particolare, per poter effettivamente definire un algoritmo come modello di machine learning, sono due le proprietà fondamentali che esso deve soddisfare: la *dipendenza dalle features* e l'*addestramento discriminativo*.
Con il primo termine si intende che ciascun documento considerato deve essere rappresentato come un vettore di caratteristiche, cioè per ogni query q il documento d sarà indicato con $x = \Phi(d, q)$, dove Φ rappresenta un estrattore di features. A seconda del problema affrontato, infatti, si possono utilizzare tantissimi valori di riferimento tramite cui rappresentare gli oggetti e da cui l'algoritmo deve prendere informazioni. Tra questi, come spiegato, spesso figurano anche i valori ottenuti dai modelli precedentemente presentati. Nel caso dei modelli parametrici, però, gli scalari variabili si considerano già studiati e fissati, così che la macchina possa concentrarsi esclusivamente sull'addestramento della migliore procedura di combinazione delle features.
L'addestramento discriminativo, invece, è ciò che era stato descritto come *modus operandi* del machine learning nel primo capitolo. In sostanza, richiedendo la presenza di questa proprietà, si vuole segnalare che ogni algoritmo di learning to rank debba avere il proprio spazio di input, spazio di output, spazio delle ipotesi e la propria funzione loss. In base a queste quattro caratteristiche, è possibile mostrare la struttura generale sottostante a

qualsiasi modello di learning to rank. Un primo aspetto che va considerato è che tutti questi metodi sottostanno al training supervisionato. Ciò significa che, nel dataset, ai documenti è sempre associata una label da cui l'algoritmo capisce l'indice di rilevanza corrispondente. In particolare, il dataset di partenza dei dati di training e validation conterrà n query q_i, $i = 1, \ldots, n$, a ciascuna delle quali saranno associati $m^{(i)}$ documenti, rappresentati come vettori di features $x^{(i)} = \{x_j^{(i)}\}_{j=1}^{m^{(i)}}$, ed il loro ordinamento reale $y^{(i)}$. Allo stesso modo, ma senza passare alla macchina l'ordinamento corretto, verrà costruito anche il test set.

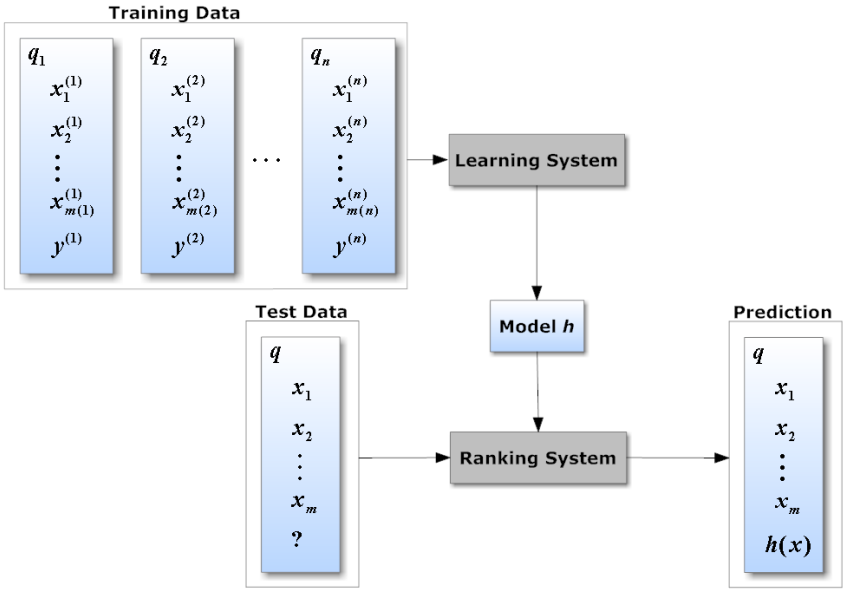


Figura 2.4: Rappresentazione dello schema generale per un algoritmo di learning to rank

A questo punto, come si vede nella figura 2.4, ogni algoritmo utilizzerà il proprio schema di apprendimento per creare un modello h in grado di effettuare predizioni sul test set, ovvero su query e documenti ancora mai visionati dalla macchina. Perciò, dopo aver eventualmente effettuato operazioni di pre-processing su tutto il dataset in modo che i dati siano opportunamente rappresentati a seconda dell'algoritmo utilizzato, le predizioni, la cui struttura dipenderà nuovamente dalla strategia adottata, permetteranno di ottenere la classifica stimata dal modello e, quindi, il risultato cercato: la risposta alla query investigata.
Prima di andare ad esplorare i singoli algoritmi di learning to rank, è possibile presentare un'importante suddivisione di tali modelli incentrata sull'approccio con cui si costruiscono gli spazi di input, di output e delle ipotesi e a seconda della tipologia di funzione loss che viene utilizzata. I tre approcci prendono il nome di approccio puntuale, a coppie e a lista.

2.3.1 Approccio Puntuale (Pointwise Approach)

Questo metodo è stato il primo ad essere utilizzato. Esso, infatti, come suggerisce il nome, prevede di considerare ogni documento come se fosse a sé. Il dataset fornito, quindi, non ha bisogno di essere modificato preliminarmente e pertanto, per quanto riguarda i documenti, sarà costituito da uno spazio bidimensionale di dimensione $n_{doc-tot} \times n_{features}$,

mentre le relevances definiranno il singolo punteggio di ogni oggetto, quindi avranno una dimensione $n_{doc-tot} \times 1$. Di conseguenza, se le etichette reali non vengono già fornite come indici di rilevanza singoli ma come preferenze tra coppie di documenti, allora può essere complicato trovare i singoli punteggi, in quanto è necessario mediare per ogni documento il numero di incontri vinti e poi applicare una trasformazione per mappare il risultato in un indice di rilevanza. Se, invece, come etichette reali si hanno le classifiche di ogni query, si possono direttamente mappare le posizioni in relevances puntuali attribuendo ai primi documenti il valore massimo, per poi diminuirlo man mano che si sono scorsi un numero fissato di documenti.
Il modello, quindi, è costruito per leggere una riga del dataset alla volta con la relativa label, senza distinguere i documenti di una query da quelli di un'altra interrogazione. Perciò, lo spazio delle ipotesi contiene delle funzioni di punteggio f in modo tale che, visionato un nuovo documento, la macchina riesca ad attribuirgli un valore. In seguito, calcolate le predizioni del test set, risulta immediato ordinare i documenti rispetto a tali punteggi così da ottenere il ranking cercato.
Per quanto concerne la funzione loss la scelta varia a seconda di come viene interpretato l'approccio puntuale. È possibile, infatti, rimodellare il problema del ranking per poterlo risolvere con tecniche di regressione, di classificazione o di regressione ordinale e, di conseguenza, utilizzare le corrispondenti funzioni loss già esistenti, come verrà approfondito nella prossima sezione. Nonostante questo garantisca più facilità implementativa, un modello così strutturato impedisce alla loss di considerare l'interdipendenza dei documenti e la loro posizione nella classifica finale, condizione che rappresenta un'altra limitazione dell'approccio pointwise, oltre alla mancata possibilità di considerare che più documenti sono relativi alla stessa query, come appena spiegato.

2.3.2 Approccio a Coppie (Pairwise Approach)

L'approccio pairwise focalizza la sua attenzione sulla comparazione di ogni possibile coppia di documenti di una query, utilizzando poi i risultati di questi confronti per ottenere il ranking finale relativo ad ogni interrogazione. Ciascuna istanza, quindi, dovrà considerare due documenti alla volta, comportando la necessità di modificare leggermente il dataset prima di poterlo utilizzare. Analizzando la singola coppia, una soluzione, consiste nella creazione di un nuovo vettore ottenuto dalla sottrazione delle features dei documenti di partenza. Tuttavia, l'implementazione più utilizzata prevede la concatenazione dei documenti. In questo caso, dato che il numero di possibili coppie non ordinate di un insieme di n elementi è $\frac{n(n-1)}{2}$ e che tale operazione va ripetuta per tutte le n query, le quali non sono necessariamente costituite dallo stesso numero di documenti, il dataset sarà ancora uno spazio bidimensionale di dimensione $\sum_{i=1}^{n} \frac{n^{(i)}_{doc-per-query}(n^{(i)}_{doc-per-query}-1)}{2} \times n_{features}$. Allo stesso modo, la dimensione dell'insieme delle relevances sarà $\sum_{i=1}^{n} \frac{n^{(i)}_{doc-per-query}(n^{(i)}_{doc-per-query}-1)}{2} \times 1$.
Per ottenere l'etichetta della coppia (u, v), a meno che esse non siano già fornite come preferenze binarie, indicata con $I_{\{A\}}$ la funzione indicatrice che assume valore 1 se il il predicato A è verificato, 0 altrimenti, si dovrà calcolare $y_{u,v} = I_{\{l_u \geq l_v\}}$ nel caso in cui si disponga delle singole relevances l_u, oppure $y_{u,v} = I_{\{\pi_l(u) \geq l_l(v)\}}$ se si dispone della permutazione π_l che corrisponde all'ordinamento reale.
Il modello costruito, quindi, dovrà essere in grado di scorrere le righe del dataset, che in questo caso rappresentano una coppia di documenti, e attribuire a ciascuna un valore di

preferenza binaria. Per raggiungere questo scopo lo spazio delle ipotesi può essere composto sia da funzioni bivariate, in grado di assegnare direttamente l'etichetta richiesta, sia da funzioni di score identiche a quelle del caso puntuale, il cui risultato sarà poi trasformato in etichetta binaria mediante l'utilizzo della funzione indicatrice I.
Anche questo secondo approccio può essere ricondotto a tecniche di machine learning preesistenti, in particolare ai casi di classificazione binaria. Ciò significa che, come per l'approccio puntuale, è possibile utilizzare direttamente le funzioni loss proprie di questa task di apprendimento automatico anche per il learning to rank. Nonostante in questo caso vengano visualizzati più documenti alla volta, la macchina è ancora incapace di riconoscere quando due diverse istanze rispondono alla stessa query e, soprattutto, la loss non considera ancora esplicitamente l'ordinamento finale. Questo significa che permangono delle discrepanze rispetto ad un approccio ideale di risoluzione di problemi di ranking.

2.3.3 Approccio a Liste (Listwise Approach)

La filosofia listwise è la più innovativa e ricalca la strategia che un essere umano adotterebbe se il numero di documenti da ordinare non fosse spropositato e non richiedesse la potenza di calcolo propria di una macchina. Come suggerisce il nome, infatti, questa strategia affronta il problema del ranking visionando contemporaneamente tutti i documenti relativi alla query e, in seguito, restituendo la lista ordinata per rilevanza dei documenti, senza passare attraverso punteggi, puntuali o binari, intermedi. L'obiettivo del ranking, difatti, consiste nella creazione di una classifica di oggetti. Quindi, ciò su cui va posta l'attenzione è la rilevanza relativa rispetto a tutti gli altri documenti piuttosto che sul singolo punteggio. Pertanto, da un punto di vista teorico, l'approccio a liste corrisponde al metodo risolutivo più diretto e più corretto, nonostante nella pratica siano poi necessari alcuni accorgimenti implementativi.
Poiché i documenti di ciascuna query devono essere studiati in contemporanea, è necessario sviluppare un dataset tridimensionale. In particolare, le sue dimensioni saranno $n_{query} \times n_{doc-per-query} \times n_{features}$ e ogni interrogazione verrà identificata come una singola istanza, la quale avrà dimensione $n_{doc-per-query} \times n_{features}$. La variazione della quantità di documenti raccolti da una query all'altra può comportare problemi implementativi dato che le dimensioni non sono uniformi per ogni istanza. Quindi, o si utilizzano dei *regged tensor* in grado di gestire input con dimensioni variabili, oppure si introduce l'iperparametro *slate length*. In questa variabile viene fissato un intero che andrà a sostituire il valore $n_{doc-per-query}$ nel dataset. Così facendo, ciascuna query avrà lo stesso numero di documenti. Perché ciò accada sono però necessarie altre operazioni di pre-processing. Infatti, se in una data query è presente un numero di documenti maggiore dello slate length fissato, allora sarà sufficiente selezionare casualmente, tra tutti quelli disponibili, tanti vettori quanti l'intero stabilito. Viceversa, se il quantitativo di documenti non è sufficiente, si mette in pratica una procedura definita *0-padding*: essa consiste semplicemente nell'aggiunta di un numero sufficiente di vettori, con features identicamente nulle, che verranno posti in fondo alla classifica o, equivalentemente, le cui relevances saranno pari a 0.
Per quanto riguarda le etichette, infatti, poiché si possa parlare effettivamente di metodo listwise, è necessario che esse rappresentino la permutazione ottimale π_y di ogni query. Perciò, se anche esse vengono passate come valori puntuali o a coppie, prima di tutto è necessario ottenere gli ordinamenti complessivi. Per farlo, nella permutazione il documento u verrà posizionato prima di v se, rispettivamente, $l_u \geq l_v$ o se $l_{u,v} = 1$. Di conseguenza,

lo spazio delle ipotesi comprende funzioni multivariate in grado di predire direttamente l'ordinamento.
In realtà, poiché può risultare molto complicata un'implementazione di questo tipo, può capitare che vengano utilizzate funzioni loss listwise con approcci pointwise o, meno comunemente, pairwise. In questi casi il dataset rimane quello appena descritto, però le relevances saranno costituite da punteggi di rilevanza singoli che verranno visualizzati query per query, cioè saranno anch'esse tridimensionali con dimensione $n_{query} \times n_{doc-per-query} \times 1$. Pertanto, anche lo spazio delle ipotesi sarà modificato e sarà costituito da funzioni del tipo $h(x) = \text{sort} \circ f(x)$. Ciò significa che prima viene utilizzata una funzione f, analoga agli approcci precedenti, in grado di prevedere i punteggi e, in seguito, verrà effettuata la composizione con una funzione di ordinamento così da ottenere il ranking complessivo.
Per quanto riguarda le funzioni loss utilizzate nell'approccio listwise, per come il modello è stato progettato, il vantaggio risiede nella possibilità di utilizzare direttamente le metriche di valutazione descritte in precedenza come valori loss da ottimizzare. Tuttavia, poiché, come detto, esse presentano problemi legati al calcolo del gradiente, sono state ideate diverse funzioni loss listwise in grado di appoggiarsi a tali metriche, così da poter considerare tutti i documenti contemporaneamente, ma, al tempo stesso, grazie a certe approssimazioni, di essere dotate anche di continuità e differenziabilità, in modo da velocizzare il calcolo del gradiente e, conseguentemente, migliorare l'ottimizzazione.

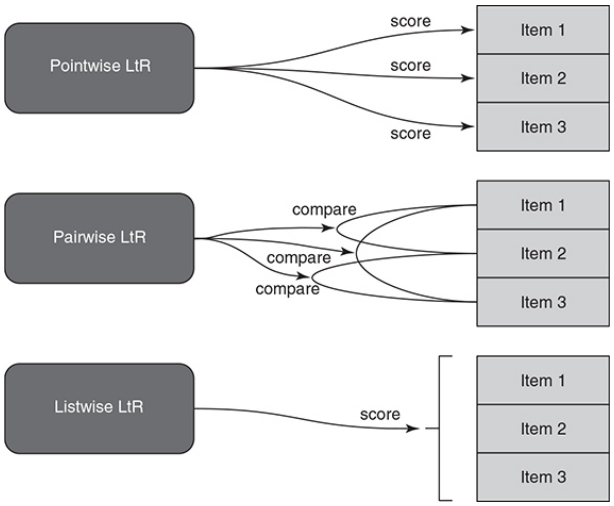


Figura 2.5: Rappresentazione grafica del funzionamento generale dei tre approcci di learning to rank presentati

Nonostante gli output intermedi siano differenti nei tre casi analizzati, il risultato finale è sempre costituito da un ordinamento di documenti, un ranking appunto. Ciò rende possibile l'utilizzo della stessa metrica di valutazione nel confronto tra diversi modelli. Quindi, poiché le metriche di valutazione sono le stesse, l'attenzione sarà ora posta, per ogni approccio, su alcuni specifici algoritmi.

2.4 Algoritmi Pointwise

Gli algoritmi che seguono l'approccio pointwise sono stati i primi ad essere implementati. La filosofia che sta alla base di questo filone di procedure prevede la riformulazione dei

problemi di ranking in situazioni, già conosciute, di machine learning, così che sia possibile adattare gli algoritmi già esistenti anche a questo ambito di ricerca. È per raggiungere questo scopo, quindi, che i modelli pointwise si focalizzano sulla predizione del singolo punteggio di rilevanza, nonostante ciò non sia esplicitamente richiesto nei problemi di ranking.
Come già accennato in precedenza, in base alla diversa strategia di machine learning che si decide di adottare, l'approccio pointwise può essere interpretato come un problema di regressione, di classificazione o di regressione ordinale, a seconda che l'output ottenuto sia, rispettivamente, un valore reale, un intero corrispondente all'etichetta di una categoria o una lista di indici relativi a categorie ordinate. In ciascun caso gli elementi del dataset e le relative etichette sono trattati come variabili aleatorie indipendenti e identicamente distribuite (i.i.d.), ottenute da un campionamento dello spazio prodotto tra spazio di input e spazio di output.
Verranno ora presentati esempi di algoritmi per ognuna delle tre categorie proposte, prediligendo le strategie che saranno utilizzate successivamente all'interno del progetto, per fornire, quindi, un quadro generale sui pro e contro degli algoritmi pointwise. Questo schema, poi, sarà adottato anche per le altre due tipologie di approccio.

2.4.1 Algoritmo di Regressione - Mean Squared Error

Il merito di aver formulato il problema del ranking come un problema di regressione va agli studiosi Cossock e Zhang [14]. Dato il gruppo di documenti $\mathbf{x}=\{x_j\}_{j=1}^m$ associati alla query q, identificate le loro etichette reali, che indicano le possibili classi di rilevanza, con $\mathbf{y}=\{y_j\}_{j=1}^m$ e detta f la funzione di score che attribuisce un qualunque valore reale all'esempio x_j, Cossock e Zhang propongono l'ottimizzazione di una qualunque funzione loss che misuri direttamente la discrepanza tra le etichette reali e le predizioni, come ad esempio la *Mean Squared Error*, la cui formula è data da

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{2.11}$$

dove n è il numero totale di documenti.
Esistono anche altre varianti, come la *Root Mean Squared Error*, che corrisponde, semplicemente, alla radice quadrata della MSE. Ma, poiché analisi teoriche hanno mostrato che la (2.11) può essere considerata un estremo superiore della quantità 1-NDCG, si è scelto di implementare la MSE nel progetto, in modo che una sua ottimizzazione minimizzasse anche il valore 1-NDCG e, di conseguenza, la metrica NDCG valesse 1, cioè il massimo.
In realtà, poiché le etichette reali corrispondono alle classi delle categorie, ha poco senso predire con un valore reale quantitativo una variabile qualitativa. Ciò spiega come mai nel progetto questo approccio sia stato modificato. Nello specifico, avendo come label veritiere valori interi da 0 a 4, si sono utilizzati come soglie i punti medi di tali intervalli. In questo modo, prima di calcolare le metriche di valutazione, i valori ottenuti dalle predizioni, calcolati come quantità reali e non intere, sono stati trasformati nelle label discrete corrispondenti, a seconda dell'intervallo a cui appartenevano.

2.4.2 Algoritmo di Classificazione - McRank

In questa interpretazione, il ranking di documenti viene trattato direttamente come un problema di classificazione. In particolare, sarà ora analizzato l'algoritmo *McRank*, una procedura ideata nel 2007 da P. Li [25], basata su metodi Montecarlo, da cui deriva il nome, in grado di gestire la classificazione multi-classe. Si è scelto di descrivere questo algoritmo in quanto, anche nel progetto, il numero di classi totali è maggiore di 2, per la precisione, 5.
Partendo dal gruppo di documenti $\mathbf{x}=\{x_j\}_{j=1}^m$ associati alla query q e dalle loro etichette reali $\mathbf{y}=\{y_j\}_{j=1}^m$, che indicano le diverse possibili categorie, si suppone di avere già costruito, con una tecnica di machine learning, un predittore multi-classe che associ all'esempio x_j la predizione $\hat{y}_j$. Pertanto, Li propone di utilizzare come funzione loss per l'addestramento del classificatore un qualunque upper bound della funzione 0-1 che conta gli errori, cioè $L(\hat{y}_j, y_j) = I_{\{y_j \neq \hat{y}_j\}}$. Ciò significa che, in base all'upper bound stabilito, alcune funzioni loss adeguate possono essere, ad esempio, la loss esponenziale o la loss logistica. In particolare, quest'ultima funzione è stata utilizzata dai ricercatori per convertire la predizione del classificatore in un punteggio di ranking.
La funzione *logistica*, infatti, ha il grande vantaggio di mappare qualsiasi valore nell'intervallo [0,1], poiché è definita da

$$P(t) = a\frac{1 + me^{\frac{-t}{\tau}}}{1 + ne^{\frac{-t}{\tau}}} \tag{2.12}$$

dove a, m, n e τ sono dei parametri reali da settare. In questo modo, l'output della funzione logistica può essere considerato una probabilità: nello specifico, la probabilità che un documento appartenga ad una certa classe k, che verrà indicata con $P(\hat{y}_j = k)$, con k intero scelto tra 0 e K-1, dove K indica il numero totale di classi tra cui scegliere. A questo punto, a seconda di quale funzione peso $T(k)$ viene adottata per assegnare più o meno importanza a determinate classi, il punteggio di ranking $f(x_j)$ si otterrà da

$$f(x_j) = \sum_{k=0}^{K-1} T(k) \cdot P(\hat{y}_j = k) \tag{2.13}$$

Solitamente, le funzioni $T(k)$ utilizzate maggiormente sono $T(k) = k$ oppure $T(k) = 2^k$, a seconda che, rispettivamente, si voglia attribuire lo stesso peso a tutte le classi o tenere maggiormente conto delle classi rappresentate da una label più grande che, quindi, generalmente, corrispondono alle più rilevanti.
All'interno del progetto realizzato, questa strategia è stata in parte adottata nel caso della regressione ordinale, come si vedrà più in dettaglio nel prossimo paragrafo.

2.4.3 Algoritmo di Regressione ordinale - PRank

Perché possa essere implementato un algoritmo di regressione ordinale, è necessario che le classi del problema studiato possano essere rappresentate secondo un ordine. Se si pensa ai problemi di ranking finora considerati, le label sono definite da valori numerici interi, per cui è possibile adottare anche questa strategia. Tuttavia, prima di spiegare il funzionamento dell'algoritmo PRank, verrà mostrata l'implementazione alternativa della strategia di regressione ordinale utilizzata, in seguito, nell'applicazione studiata.

In questo caso, date K classi con rispettive etichette da 0 a K-1, si costruisce un sistema di machine learning, in particolare una rete neurale, che non sia più a singolo output, ma che debba prevedere K-1 valori per ogni documento. Innanzitutto, devono essere modificate anche le label puntuali degli esempi del training set. Per farlo, si associa ad ogni oggetto una lista di dimensione K-1 le cui celle possono valere esclusivamente 0 o 1. Nello specifico, alla classe denotata con k corrisponderà una lista con le prime k entrate pari a 1 e con tutte le successive poste uguali a 0. Ciò significa che alla classe 0 verrà associata la lista $[0, 0, \ldots, 0]$, alla 1 la lista $[1, 0, \ldots, 0]$ e così via fino all'ultima classe, la K-1-esima, che corrisponderà alla lista $[1, 1, \ldots, 1]$. Successivamente, il predittore restituirà, per ogni documento del test set, un'etichetta di questo tipo, i cui valori di ogni cella sono forzati ad essere compresi tra 0 e 1 dalla funzione di attivazione finale sigmoide. La funzione *sigmoide*, infatti, è una caso particolare della funzione logistica di espressione

$$P(t) = \frac{1}{1 + e^{-t}} \tag{2.14}$$

ottenuta dalla (2.12) ponendo a=1, m=0, n=1 e τ=1. Contemporaneamente, per addestrare il modello è stata utilizzata, come funzione loss, la *Binary Cross-Entropy* (BCE), tipica dei problemi binari. Essa, infatti, è definita da

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log_2(p(y_i)) + (1 - y_i) \cdot \log_2(1 - p(y_i)) \tag{2.15}$$

dove y_i assume valore 1 o 0. Per come è dunque definita, per ogni termine solo uno dei due addendi contribuisce alla sommatoria, a seconda della classe a cui l'elemento appartiene, ed il totale corrisponderà all'opposto della somma delle probabilità logaritmiche che ogni punto appartenga alla classe a cui è stato associato, il tutto scalato rispetto al numero complessivo degli esempi. Applicando la funzione sigmoide il vantaggio è dato, come spiegato in precedenza, dalla possibilità di interpretare ogni cella dell'output come una probabilità. Tuttavia, in questo caso il vantaggio risiede nel fatto che il modello è costruito per predire la probabilità di appartenenza ad una classe indipendentemente dall'appartenenza ad un'altra, visto che ci sono più output distinti da calcolare. Il valore della k-esima cella dell'output, infatti, rappresenta la probabilità che l'etichetta reale dell'esempio considerato sia maggiore di k. Quindi, per ottenere la probabilità che la label del documento predetto sia esattamente k, è necessario sottrarre dalla cella k-1 il valore contenuto nella posizione k, tranne per $P(0)$, che corrisponderà a 1 - la prima cella, e per P(K-1), rappresentata direttamente dall'ultimo valore della lista. Una volta calcolate le singole probabilità, per ottenere il punteggio di ranking diventa sufficiente valutare il valore atteso, che si ottiene utilizzando $T(k) = k$ come funzione peso nella (2.13). Infine, se dal punteggio di ranking si desidera risalire alla classe corrispondente, occorre unicamente arrotondare il risultato all'intero più vicino.
Equivalentemente, per gestire la formulazione del problema mediante la regressione ordinale, è possibile concentrare gli sforzi sulla ricerca di K opportuni valori di soglia tali che $b_0 \leq b_1 \leq \cdots \leq b_{K-1}$ e tali che permettano di attribuire ogni output del predittore alla classe ordinale corretta. Poiché in un problema di ranking questo non è necessario, in quanto, per effettuare l'ordinamento, è sufficiente il valore di ranking e non la classe esatta, è stato preferito l'approccio precedente nella risoluzione del problema del progetto. In ogni modo, poiché questo è un caso largamente studiato, si riporta ora un algoritmo

in grado di gestire il caso di regressione ordinale che utilizzi tale struttura, l'algoritmo PRank.
L'algoritmo *Perceptron based Ranking*, in breve PRank, è una procedura iterativa che ha lo scopo di trovare una direzione su cui proiettare i documenti, definita da un vettore ω, in modo tale che sia più semplice utilizzare le soglie al fine di distinguere a quale classe ogni documento appartenga. All'iterazione t l'algoritmo riceve l'istanza x_j associata alla query q e restituisce la relativa predizione $\hat{y}_j = \arg\min_k\{\omega^T x_j - b_k < 0\}$. Quest'ultima verrà allora confrontata con l'etichetta reale y_j: se esse non coincidono significa che esiste almeno una soglia k per la quale il valore di $\omega^T x_j$ è dalla parte sbagliata rispetto a b_k. Per correggere questo errore, quindi, si avvicinano tra loro $\omega^T x_j$ e b_k e tale procedimento viene ripetuto finché l'algoritmo non converge.

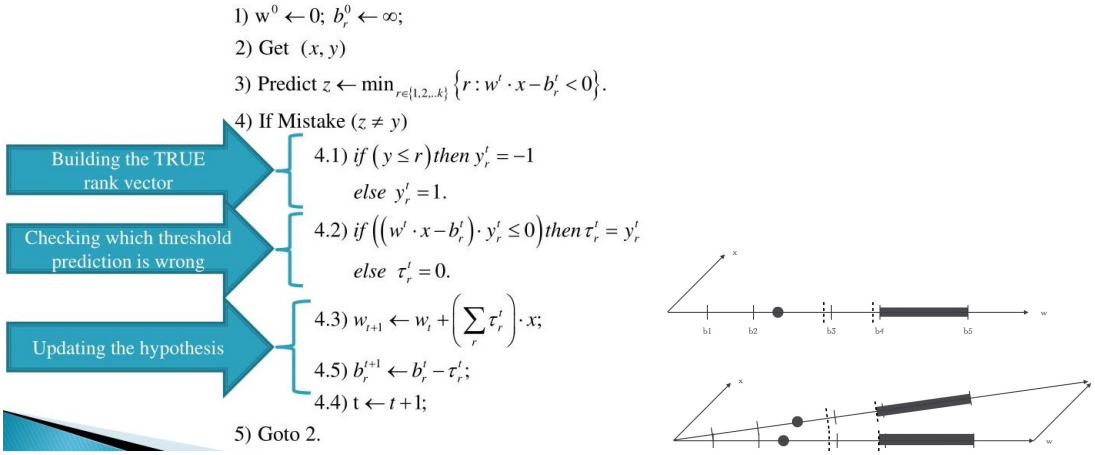


Figure 2.6a e 2.6b : A sinistra lo pseudocodice dell'algoritmo PRank, a destra un esempio grafico di iterazione in cui viene mostrata la procedura di aggiornamento

Eventualmente, esistono anche alternative a PRank che utilizzano il metodo delle SVM e le condizioni KKT per determinare soglie che possano avere margini intermedi tra di esse e che non abbiano necessariamente gli estremi coincidenti.

2.4.4 Valutazione dell'approccio pointwise

Prima dell'analisi dei vantaggi e degli svantaggi propri degli algoritmi pointwise, è bene sottolineare che, come si è potuto osservare anche dagli accorgimenti adottati nell'implementazione dei metodi del progetto, le tre sotto-categorie presentate sono, in realtà, strettamente collegate tra di loro. Tant'è che, nel caso in cui il numero di categorie K sia solamente 2, la classificazione coincide con la regressione ordinale. Pertanto, questo giustifica la possibilità di analizzare gli algoritmi pointwise globalmente, senza perdere eccessivamente di generalità.
Partendo dai pregi, va sottolineato che la facilità implementativa degli algoritmi pointwise è elevata, motivo per cui sono stati i primi ad essersi sviluppati. Inoltre è bene osservare come essi siano effettivamente algoritmi di learning to rank, al contrario di metodi come il VSM. E questo sia perché solo i documenti, non anche le query, vengono rappresentati

nello spazio delle features, sia perché tali metodi sono in grado di generalizzare dalle query del training per effettuare predizioni su interrogazioni ancora sconosciute. Ciò consente, quindi, di poter utilizzare tecniche di machine learning per velocizzare l'addestramento.
D'altro canto, gli algoritmi pointwise peccano di certe proprietà fondamentali perché possano definirsi come approcci risolutivi ottimali per i problemi di ranking. Come è già stato sottolineato, difatti, gli algoritmi puntuali si soffermano più che altro sulla ricerca della perfezione nell'individuazione del punteggio di ranking associato ad un documento. Si è già osservato, però, che l'obiettivo finale del ranking è la ricerca dell'ordine ottimale degli oggetti, che può essere raggiunto anche se tutte le etichette vengono predette in maniera errata studiando unicamente la rilevanza relativa degli esempi. Inoltre, con questa filosofia, due proprietà fondamentali che erano già richieste nelle metriche di valutazione non sono rispettate. Come prima cosa, infatti, si può notare che l'algoritmo puntuale non è in grado di distinguere a quale query sia associato un certo documento. Questa incapacità può causare problemi quando il numero di documenti è sbilanciato tra un'interrogazione e un'altra, in quanto la loss risulterà dominata dalle query a cui sono associati più documenti. In aggiunta a ciò, la posizione dei vari documenti è sconosciuta dall'algoritmo. Ma, come è già stato spiegato, le posizioni dell'ordinamento non hanno tutte la stessa importanza, poiché l'attenzione va riposta maggiormente ai primi posti. Tuttavia, non essendone a conoscenza, la loss potrebbe attribuire un'importanza eccessiva a quei documenti classificati più in basso che, quindi, sono meno importanti per l'utente.
Pertanto, queste problematiche evidenziano le necessità che hanno portato allo sviluppo degli algoritmi pairwise e listwise. Difatti, gli algoritmi pairwise, considerando le coppie di documenti, ne gestiranno più facilmente gli ordini relativi, mentre negli approcci listwise sarà gestita direttamente l'intera lista di esempi e, quindi, l'informazione relativa alla posizione degli oggetti risulterà accessibile. Proprio su questi due approcci, quindi, ci si focalizzerà nei paragrafi successivi.

2.5 Algoritmi Pairwise

In svariate applicazioni i dati che si hanno a disposizione sono sì numerosi, ma spesso disponibili solamente in maniera parziale. Può capitare, infatti, come avviene nei continui aggiornamenti che riguardano il funzionamento dei motori di ricerca, che l'unica informazione da sfruttare riguardi la preferenza dell'utente verso un documento piuttosto che verso un altro. Ciò significa che è nota solo la preferenza relativa dei documenti, non la loro rilevanza assoluta. Tuttavia, come si è potuto osservare nella sezione appena conclusa, per gli algoritmi pointwise è fondamentale essere a conoscenza dell'etichetta puntuale di ogni documento. Per questo motivo, quindi, si sono sviluppati gli algoritmi pairwise e, poiché essi si focalizzano sulla preferenza relativa tra due documenti, idealmente si avvicinano maggiormente al concetto di ranking vero e proprio.
Dovendo scegliere solo tra due alternative per ogni input, ossia se il primo documento sia più o meno rilevante del secondo, si può identificare l'approccio pairwise con una classificazione binaria. L'obiettivo, quindi, consiste nella minimizzazione di errori di predizione effettuati sulle coppie considerate. Difatti, supponendo di riuscire a non commettere alcuno sbaglio, dalle predizioni binarie si potrebbe risalire all'ordinamento globale esatto. Tuttavia, si vedrà che la ricerca della minimizzazione di questo errore potrebbe non coincidere con una delle richieste fondamentali del ranking, cioè che gli errori nelle prime

posizioni debbano pesare di più.
Va sottolineato, però, che, nonostante l'identificazione dell'approccio con un problema di classificazione binario, il metodo pairwise si differenzia notevolmente dall'approccio pointwise. Per capirne il motivo si deve prestare attenzione alla differenza delle istanze nei due metodi: mentre nel pointwise ad ogni istanza corrisponde un solo documento, in questo caso l'algoritmo lavora su due documenti alla volta. Ciò fa sì che gli algoritmi classici di classificazione non possano essere utilizzati, poiché le coppie di documenti violano l'ipotesi basilare della classificazione che ogni istanza sia indipendente dalle altre. Per questo motivo, dunque, sono nati nuovi algoritmi specifici che si fondano su strutture differenti, alcuni dei quali saranno ora descritti più dettagliatamente.

2.5.1 Ordinamento mediante una funzione di preferenza

Questo primo algoritmo fa a meno dell'utilizzo di una funzione di scoring f, limitandosi alla costruzione di una loss basata su una funzione di preferenza h. In particolare, essa dipenderà dai due documenti presenti nell'istanza e, se le etichette binarie assumono valori scelti nell'insieme $\{0, 1\}$, allora si avrà $h(x_u, x_v) = 1$ se il documento x_u verrà predetto come più rilevante di x_v, 0 nel caso opposto. Quindi, la loss associata agli esempi x_u e x_v relativi alla query q assume la forma

$$L(h; x_u, x_v, y_{u,v}) = |y_{u,v} - h(x_u, x_v)| \tag{2.16}$$

È chiaro come una loss di questo tipo si limiti esclusivamente a contare le coppie predette in maniera errata, in quanto, se l'etichetta $y_{u,v}$ e la predizione $h(x_u, x_v)$ coincidono, il valore della loss relativa a quell'istanza sarà 0, altrimenti, nel caso in cui ci sia stato un errore, si avrà 1. Quindi, mediando su tutto il training set, questa loss prenderà valori nell'intervallo [0,1], dove lo 0 indica la perfezione assoluta e l'1 che tutte le predizioni si sono verificate errate.
Essendo una funzione di preferenza in grado di fornire esclusivamente la rilevanza reciproca tra due documenti, senza calcolare un punteggio di ranking, la funzione h non può essere utilizzata direttamente per ottenere l'ordinamento finale richiesto, ma è necessario effettuare un ulteriore passaggio: deve essere risolto, quindi, il seguente problema di ottimizzazione

$$\max_{\pi} \sum_{u<v} h(x_{\pi^{-1}(u)}, x_{\pi^{-1}(v)}) \tag{2.17}$$

Ciò significa che si deve trovare la lista π che abbia il maggior numero di preferenze binarie. Tuttavia, è stato provato che il problema (2.17) è un problema NP-hard, il che sostanzialmente significa che, con una probabilità prossima a 1, risulta impossibile risolvere tale problema in tempo polinomiale rispetto alla grandezza dell'input. Quindi, poiché nel ranking si lavora con enormi quantità di dati, sono stati ideati altri algoritmi.

2.5.2 RankNet e FRank

RankNet è stato uno dei primi algoritmi ad essere utilizzato nei motori di ricerca e una sua versione modificata è adottata tutt'oggi dalla Microsoft. Al contrario dell'algoritmo precedente, in questo caso si fa uso di una funzione di scoring f e si considerano le etichette reali come probabilità, in quanto assumono valori pari a 0 o a 1. Pertanto, denotando le istanze allo stesso modo e chiamando la label che indica la rilevanza binaria $\bar{P}_{u,v}$, lo scopo

dell'algoritmo sarà la predizione della probabilità $P_{u,v}$, a valori in [0,1], che indicherà con quale possibilità il modello riterrà il documento x_u più rilevante dell'esempio x_v. Per ottenere tale probabilità il modello effettuerà il seguente calcolo basato sugli score dei documenti

$$P_{u,v}(f) = \frac{e^{f(x_u)-f(x_v)}}{1+e^{f(x_u)-f(x_v)}} \tag{2.18}$$

A questo punto come funzione loss viene riproposta la Binary Cross-Entropy definita in precedenza nella (2.15), utilizzando come y_i le $\bar{P}_{u,v}$ e, al posto di $p(y_i)$, le $P_{u,v}$ appena calcolate. Infine, gli autori originali propongono di utilizzare come modello una rete neurale che sfrutti l'algoritmo di discesa del gradiente per settare la funzione di scoring f.
Esistono però delle criticità relative a RankNet, in particolare riguardo alla loss utilizzata, motivo per il quale Tsai [44] ha proposto la variante che prende il nome di *FRank*. I problemi legati alla loss di RankNet, il cui grafico, come funzione di $f(x_u) - f(x_v)$, è rappresentato a sinistra nella figura 2.7, dipendono dal fatto che la BCE ha minimi diversi da zero e non è limitata. Ciò comporta, in primo luogo, che sia impossibile non commettere qualche errore a prescindere dal modello utilizzato, fatto che non è in accordo con le proprietà richieste da una funzione loss (fatto che vale da un punto di vista teorico per la funzione in sè, in quanto con nessuna rete neurale è possibile escludere che esistano minimi locali relativi, rispetto ai parametri da apprendere.). Come seconda conseguenza si ha che documenti con una loss molto alta possono esercitare una dominazione riguardo agli aspetti sui quali il modello debba concentrarsi, magari in quanto tali esempi sono frutto di rumore, o perché rappresentano anomalie. Tuttavia, a causa dell'elevato valore di loss ad essi associato, il modello si sofferma eccessivamente sul miglioramento di tali predizioni. Per ovviare a questo problema è stata proposta una nuova funzione, detta *fidelity loss*, descritta da

$$L(f; x_u, x_v, y_{u,v}) = 1 - \sqrt{\bar{P}_{u,v}P_{u,v}(f)} - \sqrt{(1-\bar{P}_{u,v})(1-P_{u,v}(f))} \tag{2.19}$$

ed il cui grafico è rappresentato a destra nella figura 2.7.

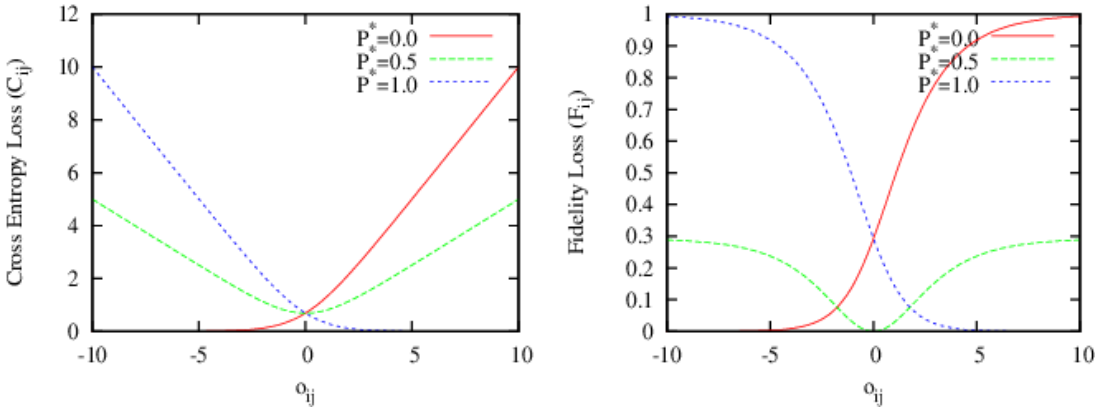


Figura 2.7: A sinistra è rappresentato il grafico della BCE, a destra quello della fidelity loss, entrambi in funzione di $f(x_u) - f(x_v)$. Sull'asse delle ascisse ci sono le predizioni, su quello delle ordinate il valore della loss e, in entrambi i casi, si considerano tre diversi valori di $\bar{P}$

Come evidenzia il confronto tra i due grafici, a differenza della BCE, la fidelity loss è limitata ad essere compresa tra 0 e 1 e, per ogni valore di $\bar{P}$, esiste almeno un minimo pari a

0, cioè il risultato che si voleva ottenere. D'altro canto, va sottolineato che la fidelity loss perde la proprietà di essere convessa, al contrario della BCE (sempre considerando la funzione da un punto di vista prettamente matematico, perché anche per la convessità, se si considera la funzione loss rispetto ai parametri che una rete neurale sta ottimizzando, essa risulta difficile da ottenere). Ciò, quindi, può creare problemi al momento dell'ottimizzazione. Tuttavia, esistono diversi risultati sperimentali che testimoniano come l'algoritmo FRank ottenga prestazioni migliori di RankNet su svariati dataset.

2.5.3 RankBoost

Il prossimo algoritmo, detto *RankBoost*, è un adattamento alla predizione di coppie di documenti della procedura *AdaBoost*. L'idea che sta alla base dell'AdaBoost, e di conseguenza di RankBoost, è "l'unione fa la forza". Difatti, in questo caso non si cerca di costruire direttamente un unico predittore forte, ma si utilizzano diversi classificatori deboli combinandoli insieme, così da ottenere un modello finale in grado di trattare anche gli esempi più critici grazie ai diversi predittori utilizzati. Il procedimento, quindi, è iterativo e ad ogni passo si utilizza un nuovo predittore che calcoli gli errori commessi nella classificazione. A seconda del risultato ottenuto, vengono poi aggiornati due diversi pesi: uno relativo agli esempi, che sarà tanto più grande tanti più errori vengono commessi su quell'esempio, in modo che ci si concentri a mettere a punto le predizioni sui dati ancora mal classificati; il secondo che riguarda i predittori, affinché, in base alle singole performance, ogni componente incida in proporzione a quanto bene si sia comportata.
In particolare, RankBoost procede minimizzando la funzione loss esponenziale

$$L(f; x_u, x_v, y_{u,v}) = e^{-y_{u,v}(f(x_u)-f(x_v))} \tag{2.20}$$

Lo pseudocodice della procedura è descritto nell'Algoritmo 1, dove D_t rappresenta la distribuzione delle coppie di documenti, f_t è il classificatore debole selezionato durante la t-esima iterazione e α_t indica il peso da attribuire al predittore corrispondente durante la combinazione lineare finale.

Algorithm 1 Algoritmo di addestramento per RankBoost

Require: distribuzione inizialeD_1 sulle coppie di documenti date in input
  for $t = 1, \ldots, T$ **do**
    Addestra classificatore debole f_t basandoti sulla distribuzioneD_1
    Scegli α_t
    Aggiorna $D_{t+1}(x_u^{(i)}, x_v^{(i)}) \leftarrow \frac{1}{Z_t} D_t(x_u^{(i)}, x_v^{(i)}) e^{\alpha_t(f_t(x_u^{(i)}) - f_t(x_v^{(i)}))}$
    dove $Z_t \leftarrow \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} D_t(x_u^{(i)}, x_v^{(i)}) e^{\alpha_t(f_t(x_u^{(i)}) - f_t(x_v^{(i)}))}$
  end for
  Output:$f(x) = \sum_t \alpha_t f_t(x)$

Per quanto riguarda la politica di aggiornamento del peso α_t esistono 3 possibilità:

- Il primo approccio è il più generico e si basa sul fatto che la quantità Z_t, vista come funzione di α_t, possiede un unico minimo. Pertanto, per trovare il peso di ogni predittore è sufficiente procedere numericamente mediante l'utilizzo di una ricerca dicotomica (più comunemente indicata con il termine inglese *binary search*).

- Questo secondo metodo è applicabile esclusivamente nel caso in cui f_t prenda valore nell'insieme $\{0, 1\}$. Con questa ipotesi, per $b \in \{-1, 0, 1\}$, si definisce

$$W_{t,b} = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} D_t\left(x_u^{(i)}, x_v^{(i)}\right) I_{f_t(x_u^{(i)})-f_t(x_v^{(i)})=b} \tag{2.21}$$

  Quindi, per minimizzare Z_t, si ottiene che α vale

$$\alpha_t = \frac{1}{2}\log\left(\frac{W_{t,-1}}{W_{t,1}}\right) \tag{2.22}$$

- Infine, se si vuole approssimare Z_t in modo che si abbia una condizione generale valida quando f_t prende valori in [0,1], è sufficiente fissare la quantità

$$r_t = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} D_t\left(x_u^{(i)}, x_v^{(i)}\right)\left(f_t(x_u^{(i)}) - f_t(x_v^{(i)})\right) \tag{2.23}$$

  In questo caso, quindi, si aggiornerà α come segue

$$\alpha_t = \frac{1}{2}\log\left(\frac{1+r_t}{1-r_t}\right) \tag{2.24}$$

Dal momento che RankBoost è una rivisitazione di AdaBoost, esso ne eredita anche le principali proprietà. Nello specifico, tre importanti vantaggi sono dati dalla mancanza di iperparametri da settare, fatta eccezione per il numero massimo di iterazioni t, dalla capacità di generalizzare, grazie alla specializzazione su tanti sottospazi dovuta ai diversi predittori, e dalla convergenza sul training set. Di contro, però, si mantengono anche gli aspetti negativi che caratterizzano AdaBoost. Quest'ultimo infatti, dipende fortemente dai dati e dallo spazio delle ipotesi entro cui vengono selezionati i predittori, e, soprattutto, è molto suscettibile agli errori: a causa del fatto che il peso degli esempi predetti erroneamente aumenta sempre di più, l'algoritmo presterà attenzione sempre crescente proprio a quei dati, anche nel caso in cui essi siano affetti da rumore, rallentando eccessivamente l'intera procedura nel caso in cui compaia una significativa quantità di outliers. Una soluzione simile, che prevede cioè il completo adattamento di una tecnica già conosciuta al caso in cui ogni istanza coincide con una coppia di documenti, è il Ranking SVM. In questa circostanza, infatti, si utilizza la stessa funzione obiettivo della *Support Vector Machine* (SVM) con le condizioni KKT, permettendo così all'algoritmo di ereditare le capacità di generalizzazione e di trattamento dei problemi non-lineari tipiche della SVM.

2.5.4 LambdaRank

Come già spiegato, gli algoritmi pairwise possono essere considerati un miglioramento dei metodi puntuali, pur continuando ad attribuire ad ogni documento la medesima importanza durante la fase di addestramento anziché soffermarsi maggiormente sugli esempi che occupano le prime posizioni, considerazione che invece le metriche come la NDCG riescono a trattare. Il problema risiede nel fatto che l'informazione relativa alla posizione è disponibile solamente dopo aver effettuato l'ordinamento, che però è un'operazione non

differenziabile. Difatti, anche se, dopo aver ordinato, la concentrazione è rivolta alle prime posizioni, alla base di ciò c'è comunque una procedura di ranking, che necessariamente dipende da tutti i documenti del dataset, che può risultare anche davvero grande. Ciò, però, significa che rispetto alle metriche di valutazione considerate, un approccio come quello di RankNet non risulta essere ottimale. È per questo motivo, quindi, che, dopo aver proposto un esempio che chiarisca la situazione, verrà presentato un algoritmo basato su un approccio completamente differente.

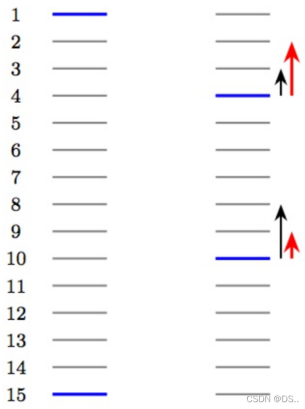


Figura 2.8: Due possibili esempi di ordinamento con gli stessi documenti. In blu sono evidenziati quelli rilevanti, in grigio quelli non significativi. Le frecce laterali indicano i gradienti: rispettivamente, quella nera è relativa a RankNet, la rossa è ciò che si desidera avere.

Nell'esempio rappresentato nella figura 2.8 sono proposti due possibili ordinamenti di documenti, di cui 2 sono rilevanti, quelli evidenziati di blu, mentre gli altri non sono importanti. Se si considera il numero di errori di coppie predette, a sinistra esso è pari a 13, mentre a destra vale 11. Ciò significa che un algoritmo come RankNet, che punta a minimizzare questo numero di errori, predilige la situazione di destra. Tuttavia, se si calcola il valore della NDCG nelle due situazioni, si vedrà che esso risulterà maggiore per la situazione di sinistra, in quanto in quel caso il documento in prima posizione è classificato correttamente. Quindi, poiché si vuole dare importanza agli esempi posizionati nelle prime fasce e la NDCG è una metrica che tiene conto di questa proprietà, è preferibile, tra le due, ottenere la situazione raffigurata a sinistra. Ciò significa che, rappresentando con delle frecce i gradienti, si vuole cercare di ottenere un algoritmo basato su una loss che abbia un gradiente dal comportamento analogo a quello rappresentato dalla freccia di colore rosso, piuttosto che da quella di colore nero, che peraltro coincide con il gradiente relativo a RankNet. Perciò, seppur RankNet possa consentire di ottenere prestazioni soddisfacenti, soprattutto se dotato di opportuni accorgimenti come, un adeguato criterio di cessazione dell'algoritmo incentrato sul valore che la loss assume sul validation set, è possibile raggiungere risultati ancora migliori con l'algoritmo LambdaRank.
L'idea che sta alla base di *LambdaRank* è che sia molto più semplice scrivere direttamente un gradiente che, dopo aver ottenuto un primo ordinamento, abbia le proprietà desiderate per migliorare quella specifica classifica, piuttosto che trovare una generica funzione loss differenziabile che riesca a trattare qualsiasi tipo di situazione. In questo modo, potendosi soffermare sul preciso ordinamento considerato, si costruisce a priori il gradiente desiderato, il quale si fonderà su un'implicita funzione loss C, poiché è il gradiente la quantità che

serve all'algoritmo durante la fase di ottimizzazione. In particolare, come si è visto dall'esempio proposto, la proprietà che LambdaRank ricerca per il gradiente della funzione costo C, è

$$|\frac{\partial C}{\partial s_{j_i}}| \gg |\frac{\partial C}{\partial s_{j_k}}| \tag{2.25}$$

dove s_{j_k} è il punteggio di ranking relativo al documento j_k e $s_{j_k} \gg s_{j_i}$. Quindi, non dovendo costruire una funzione loss perfetta e concentrandosi solo sulle proprietà che si richiedono al gradiente, LambdaRank può essere considerato un metodo semplice che, al contempo, si è dimostrato essere in grado di sfruttare funzioni di costo non lisce e capace di velocizzare il tempo di addestramento richiesto da RankNet.

2.5.5 Valutazione dell'approccio pairwise

Una prima osservazione fondamentale riguarda l'adattamento dei metodi pairwise al caso della classificazione multipla. Ciò significa che le etichette possono assumere più di due valori, descrivendo così anche il grado di rilevanza di un documento. Come visto, però, gli algoritmi pairwise sono in grado di gestire una preferenza relativa esclusivamente binaria, causando in questo modo la perdita dell'informazione corrispondente al livello di significatività di un esempio. Per ovviare a questo problema Qin [36] ha proposto un algoritmo denominato *multiple hyperplane ranker* (MHR) che può essere applicato a qualsiasi procedura pairwise. L'idea su cui basa questo algoritmo è detta *divide et impera*, un approccio comune in informatica che prevede di semplificare il problema in tanti sotto-problemi minori e più semplici, facendo confluire, successivamente, le singole soluzioni a quella globale. Ipotizzando la presenza di K differenti categorie, si devono addestrare $\frac{K(K-1)}{2}$ classificatori classici, in modo tale che ognuno si concentri sulla distinzione di sole due categorie alla volta. In seguito, ciascuna predizione si mescola alle altre per produrre il risultato finale. In particolare, detto $f_{k,l}$ il modello addestrato sulle categorie k ed l, il predittore finale per il ranking globale si ottiene con il *Metodo Borda*, che consiste nel calcolare la seguente somma pesata

$$f(x) = \sum_{k,l} \alpha_{k,l} f_{k,l}(x) \tag{2.26}$$

dove $\alpha_{k,l}$ può essere fissato a priori o ottenuto mediante un secondo addestramento compiuto su un altro validation set. L'accorgimento proposto da Qin con l'algoritmo MHR ha permesso di incrementare significativamente i risultati ottenuti tramite algoritmi di pairwise classici.
Nonostante gli algoritmi pairwise godano di svariati benefici rispetto agli approcci pointwise, dovuti soprattutto alla capacità di intravedere le relazioni presenti tra i documenti, nemmeno questi si rivelano essere in grado di tenere in considerazione la posizione degli esempi nell'ordinamento così da scalare il peso di ciascun oggetto con l'aumentare delle posizioni in classifica, ad eccezione, in parte, di LambdaRank.
Ciononostante, questo non è lo scoglio più grande da affrontare quando si utilizzano gli algoritmi pairwise. Esso dipende piuttosto dall'aumento della complessità computazionale dovuto all'ingrandimento del dataset. Infatti, dovendo costruire coppie di documenti all'interno di ogni query, partendo da k documenti si arriva a dover trattare $\frac{k(k-1)}{2} \sim k^2$ istanze e, considerando che in questo tipo di applicazioni i dataset di partenza sono solitamente già molto grandi, questo incremento quadratico può rendere intrattabile il problema, come

si vedrà anche nello sviluppo del progetto. Non solo: la relazione quadratica può provocare anche un importante sbilanciamento di documenti tra una query e l'altra, in quanto anche una piccola differenza di documenti puntuali può causare un eccessivo divario tra le istanze a coppie. In questo caso, però, esiste una soluzione: la normalizzazione, per ogni query, della loss pairwise rispetto al numero di coppie di documenti relative alle rispettive interrogazioni. In questo modo ogni loss assume un ordine di grandezza equiparabile a quelli di tutte le altre funzioni costo, ottenendo così performance migliori nell'addestramento.
Non rimane altro, ora, che investigare gli algoritmi di più recente innovazione e, probabilmente, quelli che più riflettono l'approccio corretto ad un problema di ranking, gli algoritmi listwise.

2.6 Algoritmi Listwise

Gli algoritmi listwise sono stati introdotti perché riflettono un approccio completamente differente rispetto ai due metodi precedenti. Difatti, essi non cercano di ricondurre il problema a casi di classificazione o di regressione, ma, piuttosto, adottano due diverse strategie, a seconda della sotto-categoria a cui appartengono. Una prima strada consiste nell'ottimizzazione diretta delle metriche di valutazione spiegate in precedenza: in questo caso, l'output dell'algoritmo contiene tutti i punteggi di rilevanza dei documenti associati alla query studiata e la funzione loss è costruita come un'approssimazione o una limitazione della metrica selezionata. Nella seconda opzione, invece, il modello è costruito per fornire come predizione direttamente l'ordine dei documenti dell'interrogazione, così che la loss possa misurare la differenza tra la permutazione proposta e quella corretta. Intuitivamente, quindi, questi algoritmi dovrebbero essere in grado di ottenere prestazioni migliori rispetto ai precedenti, poiché affrontano frontalmente il problema del ranking.

2.6.1 Algoritmo che ottimizza le metriche di valutazione - SoftRank

Tra le metriche di valutazione disponibili, la NDCG è certamente la più utilizzata e la più significativa. Per questo motivo, quindi, nella sezione 2.4.1 si era già proposto di utilizzare la quantità 1-NDCG come funzione obiettivo da minimizzare. Tuttavia, per implementare la suddetta funzione, è necessario calcolare il fattore di sconto D_k e tale quantità può essere ottenuta solamente dopo aver completato un intero ordinamento, operazione che, come già spiegato, non è differenziabile. Una prima soluzione la si può trovare in LambdaRank, algoritmo descritto nel paragrafo 2.5.4. Esso è stato classificato come metodo pairwise, ma l'utilizzo iterativo della NDCG nel bilanciamento delle istanze lo può far considerare come una via di mezzo tra un approccio listwise e uno pairwise. Ad esclusione di questo caso particolare, la non-differenziabilità della NDCG fa capire il motivo per cui si debbano costruire approssimazioni o limitazioni di tale metrica.
Un esempio, in questo senso, si può trovare nell'algoritmo *SoftRank*. La soluzione proposta consiste nell'introdurre casualità ai punteggi di rilevanza per utilizzare l'aspettazione della NDCG come approssimazione della metrica stessa. Per raggiungere questo scopo, inizialmente, SoftRank definisce una distribuzione dei punteggi: data la lista di documenti $\mathbf{x}=\{x_j\}_{j=1}^{m}$ associati alla query q non viene più predetto il punteggio $s_j = f(x)$ come un

valore deterministico, ma esso viene trattato come una variabile aleatoria. Pertanto, la predizione fornirà uno score probabilistico $s_j \sim N(f(x_j), \sigma_s^2)$, che sarà dominato, quindi, da una distribuzione Gaussiana di media $f(x_j)$ e varianza σ_s^2.
Dopo aver completato questa operazione, l'attenzione passa alla distribuzione dei posizionamenti. Grazie alla casualità introdotta nei punteggi, le posizioni non saranno più univoche rispetto ai documenti ma, al contrario, a ciascun esempio potrà essere associata ogni posizione. In particolare, la probabilità che un documento sia classificato prima di un altro sarà data da

$$p_{u,v} = \int_0^{+\infty} N(f(x_u) - f(x_v), 2\sigma_s^2)\, ds \tag{2.27}$$

probabilità che ora si possono calcolare grazie alla regolarità introdotta nella nuova funzione dalla casualità. Basandosi su queste formule, quindi, la distribuzione dei posizionamenti può essere calcolata in maniera iterativa: infatti, supponendo di aver già classificato il documento x_j, nel momento in cui si deve posizionare l'elemento x_u, se quest'ultimo può scavalcare x_j allora la posizione di x_j verrà incrementata di uno, altrimenti resterà invariata. In generale, la probabilità $p_j(r)$ che x_j sia classificato alla r-esima posizione può essere ottenuta da

$$p_j^u(r) = p_j^{(u-1)}(r-1)p_{u,j} + p_j^{(u-1)}(r)(1 - p_{u,j}) \tag{2.28}$$

Infine, ottenuta la distribuzione delle posizioni, SoftRank calcola l'aspettazione della NDCG@m, dove m indica il numero di documenti associati alla query. Quest'ultima funzione, che prende il nome di *SoftNDCG*, sarà la funzione obiettivo che andrà massimizzata dall'algoritmo e sarà descritta da

$$\text{SoftNDCG} := \frac{1}{Z_m} \sum_{j=1}^{m} (2^{y_j} - 1) \sum_{r=0}^{m-1} \eta(r) p_j(r) \tag{2.29}$$

Pertanto, utilizzando come modello una rete neurale, la funzione loss da minimizzare sarà data da 1-SoftNDCG e si potrà decidere se pre-stabilire la varianza σ_s o considerarla un parametro da addestrare.

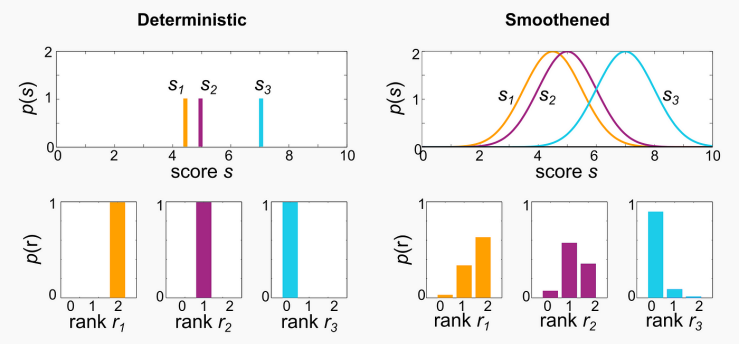


Figura 2.9: Rappresentazione grafica di come la casualità introdotta da SoftRank consenta di ottenere una funzione loss liscia

Un'altra strada che permette di ottimizzare le metriche di valutazione si fonda sull'utilizzo delle SVM. Poiché l'attenzione di questo elaborato è rivolta soprattutto ai metodi

implementabili mediante l'utilizzo di una rete neurale, ne verrà data solamente una veloce spiegazione. In questo caso, l'idea consiste nell'inserimento della misura di valutazione che si vuole ottimizzare all'interno dei vincoli del metodo SVM. Tuttavia, poiché il numero di etichette errate può essere esponenziale nel numero dei documenti di partenza, ciò che si fa è considerare esclusivamente quei vincoli che sono stati violati in maniera più significativa e l'ottimizzazione si riduce ad essi. Diventa cruciale, quindi, individuare algoritmi in grado di selezionare tali vincoli in maniera efficiente, procedura che è stata dimostrata poter essere eseguita con un costo computazionale pari a $O(m \log m)$, dove m indica il numero di documenti associati ad una query.

2.6.2 Algoritmo di boosting sulle metriche di valutazione - AdaRank

Xu e Li [48] hanno proposto una variante listwise dell'algoritmo RankBoost descritto nella sezione 2.5.3 in grado di sfruttare direttamente le metriche di valutazione. Al contrario dell'algoritmo appena descritto, però, questo non richiede che tali metriche siano continue e differenziabili. La strategia, quindi, si fonda nuovamente sull'utilizzo del boosting: si dovrà utilizzare, pertanto, un algoritmo iterativo in grado di aggiornare la distribuzione delle query per poter poi calcolare il coefficiente-peso da utilizzare nella combinazione lineare conclusiva. La differenza con RankBoost sta nel fatto che, anziché sfruttare la loss esponenziale, in *AdaRank* si fa uso direttamente delle metriche di valutazione. Il flusso dell'algoritmo, quindi, è il seguente

Algorithm 2 Algoritmo di addestramento per AdaRank

Require: distribuzione inizialeD_1 del gruppo di documenti delle query in input
  for $t = 1, \dots, T$ **do**
    Addestra classificatore debole $f_t(\cdot)$ basandoti sulla distribuzioneD_t
    Scegli $\alpha_t \leftarrow \frac{1}{2} \log \frac{\sum_{i=1}^{n} D_t(i)(1+M(f_t,\mathbf{x}^{(i)},\mathbf{y}^{(i)}))}{\sum_{i=1}^{n} D_t(i)(1-M(f_t,\mathbf{x}^{(i)},\mathbf{y}^{(i)}))}$
    Aggiorna $D_{t+1}(i) \leftarrow \frac{e^{-M(\sum_{s=1}^{t} \alpha_s f_s,\mathbf{x}^{(i)},\mathbf{y}^{(i)})}}{\sum_{j=1}^{n} e^{-M(\sum_{s=1}^{t} \alpha_s f_s,\mathbf{x}^{(j)},\mathbf{y}^{(j)})}}$
  end for
  Output: $\sum_t \alpha_t f_t(\cdot)$

Grazie all'utilizzo del boosting, anche AdaRank è in grado, come RankBoost, di focalizzarsi sulle interrogazioni più difficoltose da classificare. Tuttavia, si è dimostrato che, perché ci sia convergenza, è necessario che la quantità $|M(\sum_{s=1}^{t} \alpha_s f_s, \mathbf{x}, \mathbf{y}) - M(\sum_{s=1}^{t-1} \alpha_s f_s, \mathbf{x}, \mathbf{y}) - \alpha_t M(f_t, \mathbf{x}, \mathbf{y})|$ sia molto piccola, ma questo implica che la metrica di valutazione M, come funzione di f_t, sia lineare. Ciononostante, nella pratica questo potrebbe non accadere. Di conseguenza, AdaRank potrebbe non convergere e ciò renderebbe necessario aggiungere criteri di stop al processo.

2.6.3 Algoritmo di minimizzazione di una loss listwise - ListNet

In questa seconda categoria di algoritmi si adotta una funzione loss in grado di misurare la discrepanza tra l'ordinamento proposto dalla macchina e quello reale. Nonostante in questo modo non si ottimizzino direttamente le metriche di valutazione, la capacità di

confrontare, con un unico sguardo, entrambe le classifiche permette a questi di algoritmi di raggiungere comunque performance più che soddisfacenti.
Un primo esempio, in questo senso, è dato da *ListNet*, un algoritmo progettato da Zhe Cao nel 2007 [9]. L'idea di ListNet è un'estensione al caso listwise del procedimento pairwise adottato da RankNet e spiegato nel paragrafo 2.5.2: anziché minimizzare le predizioni errate sulle coppie si cerca ora di ottimizzare una loss listwise che misuri direttamente gli errori relativi a tutta la lista di documenti. Per raggiungere questo scopo si sfruttano le distribuzioni di probabilità definite sullo spazio delle permutazioni. Grazie allo stretto legame che intercorre tra una lista di documenti ed una permutazione e dato che la teoria delle permutazioni è già stata largamente studiata, si sfrutterà il modello di Plackett-Luce per implementare l'algoritmo.
Difatti, supponendo di denotare i documenti da classificare con indici interi che vanno da 1 a n, una permutazione π è una biezione da $\{1, 2, \dots, n\}$ in se stessa, dove con $\pi^{-1}(j)$ verrà indicato il documento classificato alla posizione j e con Ω_n l'insieme di tutte le possibili permutazioni composte da n elementi. Supposto che esista una funzione f di scoring che assegni i punteggi $\mathbf{s}=\{s_j\}_{j=1}^n$ agli n oggetti della lista, dove $s_j = f(x_j)$, si assume che tale funzione crei incertezza nelle predizioni: ciò significa che ogni permutazione è possibile. Quindi, il modello di Plackett-Luce, sfruttando la regola della catena, definisce la probabilità di esistenza di ogni permutazione come segue

$$P(\pi|\mathbf{s}) = \prod_{j=1}^{n} \frac{\varphi\left(s_{\pi^{-1}(j)}\right)}{\sum_{u=j}^{n} \varphi\left(s_{\pi^{-1}(u)}\right)} \tag{2.30}$$

dove φ è una qualunque funzione di trasformazione che sia crescente e strettamente positiva, ad esempio una funzione lineare, esponenziale o sigmoide. Tale rappresentazione possiede proprietà vantaggiose che le consentono di descrivere appropriatamente, data la funzione di scoring, la verosimiglianza di una permutazione. Infatti, come dimostrato in [9], valgono i seguenti risultati

Lemma 2.1. *Le probabilità di permutazione $P(\pi|s)$, $\pi \in \Omega_n$, determinano una distribuzione di probabilità sull'insieme delle permutazioni, cioè, $\forall \pi \in \Omega_n$, si ha che $P(\pi|s) > 0$ e $\sum_{\pi\in\Omega_n} P(\pi|s) = 1$.*

Teorema 2.2. *Date due permutazioni π e $\pi' \in \Omega_n$, se*

1. $\pi^{-1}(p) = \pi'^{-1}(q)$, $\pi^{-1}(q) = \pi'^{-1}(p)$, $p < q$;
2. $\pi^{-1}(r) = \pi'^{-1}(r)$, $r \neq p, q$;
3. $s_{\pi^{-1}(p)} > s_{\pi^{-1}(q)}$,

allora $P(\pi|s) > P(\pi'|s)$

Teorema 2.3. *Per n oggetti, se $s_1 > s_2 > \cdots > s_n$, allora $P(\{1, 2, \dots, n\}|s)$ è la probabilità di permutazione più alta e $P(\{n, n-1, \dots, 1\}|s)$ è la probabilità di permutazione più bassa tra tutte le probabilità di permutazioni di n oggetti.*

In sostanza, il lemma permette di trattare effettivamente le probabilità definite nella (2.30) come distribuzioni di probabilità. In aggiunta, il Teorema 2.2 conferma ciò che l'intuizione suggerisce, cioè che, dopo aver effettuato uno scambio tra due documenti, dove

quello con un punteggio maggiore viene abbassato di posizione a favore di un oggetto con score minore, la probabilità relativa alla nuova permutazione diminuisce rispetto all'originale. Infine, l'ultimo teorema spiega che, seppur tutte le permutazioni siano teoricamente possibili, quella che rispecchia l'ordine decrescente stabilito dai punteggi assegnati dalla funzione f ha la probabilità maggiore di manifestarsi. Detto ciò, è possibile dunque affermare che i fattori $\frac{\varphi(s_{\pi^{-1}(j)})}{\sum_{u=j}^{n}\varphi(s_{\pi^{-1}(u)})}$ che costituiscono la produttoria della (2.30) sono delle probabilità condizionali, come è chiaramente osservabile nel prossimo esempio.
Siano A, B e C i tre documenti associati alla query q che si vogliono classificare. Allora, la probabilità che si selezioni la permutazione $\pi = (A, B, C)$ è definita da $P_\pi = P_1 P_2 P_3$, cioè è determinata dal prodotto delle tre seguenti quantità:

P_1 : Indica la probabilità che il documento A sia classificato al primo posto nella permutazione π. Per ottenerla, quindi, si deve comparare il punteggio di A con quello di tutti gli altri documenti, perciò

$$P_1 = \frac{\varphi(s_A)}{\varphi(s_A) + \varphi(s_B) + \varphi(s_C)} \tag{2.31}$$

P_2 : È la probabilità condizionale che il documento B venga posizionato come secondo, dato che A è già stato fissato come primo. Pertanto, per ottenere P_2 si deve comprare il punteggio di B con quello di tutti gli altri eccetto A, ossia, poiché in questo caso l'unico altro documento è C,

$$P_2 = \frac{\varphi(s_B)}{\varphi(s_B) + \varphi(s_C)} \tag{2.32}$$

P_3 : Corrisponde alla probabilità condizionale che il documento C compaia in terza posizione, dato che A e B sono stati indicati, rispettivamente, primi e secondi della lista. In questo esempio, dal momento che non compare nessun altro elemento oltre a C, si ha semplicemente che $P_3 = 1$

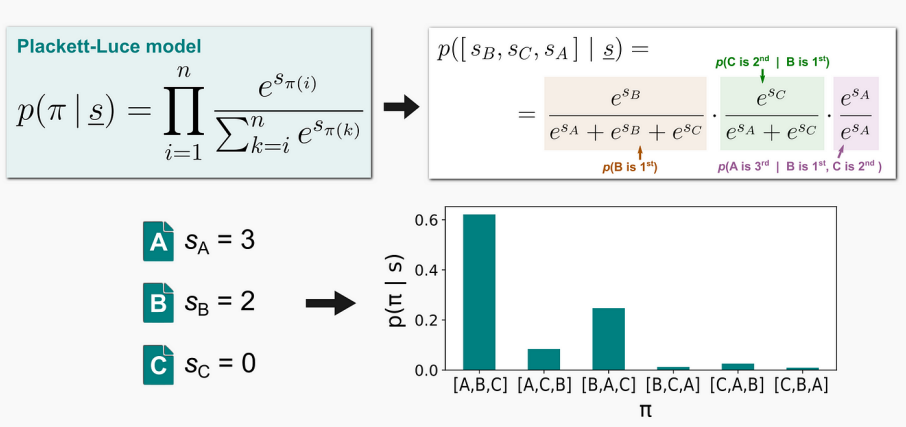


Figura 2.10: Modello di Plackett-Luce nel caso in cui la funzione φ sia esponenziale e rappresentazione grafica di un esempio di classificazione di tre documenti

Definita una prima distribuzione di probabilità basata sul modello Plackett-Luce, la costruzione della funzione loss usata nell'algoritmo ListNet necessita di una seconda distribuzione $P_y(\pi)$, che stavolta si baserà sulle etichette reali. Perché funzioni, esse devono essere passate come punteggi di rilevanza, in accordo con l'output della funzione di scoring da cui costruire la prima distribuzione. Ciò significa che, se le etichette vengono date come lista correttamente classificata, è necessario mappare preliminarmente tali posizioni in valori reali. Successivamente, dati i due modelli aleatori appena costruiti, si implementa come funzione loss listwise la divergenza di Kullback–Leibler D_{KL} calcolata proprio tra le due distribuzioni. Essa, quindi, assume la seguente forma

$$\begin{aligned} L(f; \mathbf{x}, \pi_y) &= D(P(\pi|\varphi(f(w, \mathbf{x})))||P_y(\pi)) \\ &= \sum_{\pi \in \Omega_n} P(\pi|\varphi(f(w, \mathbf{x}))) \log_2 \left(\frac{P(\pi|\varphi(f(w, \mathbf{x})))}{P_y(\pi)} \right) \end{aligned} \tag{2.33}$$

Per utilizzare ListNet si costruisce solitamente un modello di rete neurale, il quale sfrutta i metodi di discesa del gradiente per minimizzare la loss (2.33). Lo pseudocodice corrispondente è descritto nell'Algoritmo 3

Algorithm 3 Algoritmo di addestramento di ListNet

Require: dati del training set$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
  Parametri: numero di iterazioni T e learning rate η
  Inizializzare il parametro ω
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, m$ **do**
      Dai $x^{(i)}$, relativo alla query $q^{(i)}$ in input, alla rete neurale
      Calcola la lista degli score $z^{(i)}(f_\omega)$ utilizzando ω corrente.
      Calcola il gradiente $\Delta\omega$
      Aggiorna $\omega \leftarrow \omega - \eta\Delta\omega$
    end for
  end for
  Output: modello ω

dove il gradiente $\Delta\omega$ è calcolato come derivata parziale della loss L rispetto al modello ω, cioè è dato da $\Delta\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial\omega}$, e, dal momento che si utilizza una rete neurale, si ha che $f_\omega(x_j^{(i)}) = \langle\omega, x_j^{(i)}\rangle + b$.
Analizzando l'algoritmo ListNet da un punto di vista computazionale, emergono dei problemi durante la fase di addestramento. Le criticità derivano dalla necessità della loss di utilizzare tutte le possibili permutazioni relative a ciascuna query. Ma, se il numero di documenti relativi ad una certa interrogazione è pari a n, il numero di permutazioni è $n!$ e questo rende il problema intrattabile visto che il fattoriale cresce ancora più velocemente dell'esponenziale. Per ovviare alla questione, si è proposto un nuovo modello di Plackett-Luce, detto modello *top-k*, da cui ricavare la loss dipendente dalla divergenza di Kullback-Leibler. In particolare, esso consiste nel considerare la probabilità che un oggetto possa essere classificato esclusivamente nelle prime k posizioni. Per ottenere tali valori è sufficiente porre l'estremo superiore della produttoria della (2.30) pari proprio a k. In questo modo si può dimostrare che queste probabilità formano comunque una distri-

buzione di probabilità e che il loro utilizzo riduce la complessità del problema all'ordine polinomiale.

2.6.4 Algoritmo di minimizzazione di una loss listwise - ListMLE

Nonostante l'implementazione del modello top-k abbia ridotto il costo computazionale necessario per utilizzare ListNet, questa introduzione ha imposto un compromesso problematico. Difatti, se il valore di k non è sufficientemente piccolo, il costo computazionale richiesto dall'algoritmo rimane comunque elevato. D'altro canto, più k è piccolo maggiore è la perdita di informazione riguardo alla permutazione e ciò comporta una riduzione nell'efficacia della procedura. Per risolvere questo problema, nel 2008 è stato concepito un nuovo algoritmo, denominato *ListMLE*.
Proposto da Xia, Liu, J. Wang, W. Wang e Li [47], ListMLE si basa ancora una volta sul modello Plackett-Luce delle permutazioni, sfruttando però una differente funzione loss. Tale scelta è stata effettuata solamente dopo aver effettuato approfonditi studi teorici sulle proprietà di varie funzioni loss candidate, studi descritti dettagliatamente in [47]. In particolare, si sono studiate la *likelihood loss*, la *cosine loss* e la *cross-entropy loss*. Adottando in tutte e tre le soluzioni la probabilità del modello Plackett-Luce, ossia utilizzando la (2.30) come valore della P, la prima si basa, in realtà, sul logaritmo negativo della verosimiglianza della probabilità, e quindi è descritta da

$$L = -\log P(\pi|\mathbf{s}) \tag{2.34}$$

La seconda, invece, è incentrata sulla cosine similarity, perciò sarà data da

$$L = \frac{1}{2}\left(1 - \frac{g(x)^T f(x)}{\|g(x)\|\|f(x)\|}\right) \tag{2.35}$$

dove g è la funzione che mappa le posizioni reali negli score reali e f è la funzione che predice i punteggi.
Infine, l'ultima coincide con la loss utilizzata in ListNet, quindi la sua formula si trova nella (2.33).
Per quanto riguarda le proprietà di cui ogni funzione gode, è possibile vedere come la più completa sia la funzione di verosimiglianza. Difatti, tutte le loss considerate sono consistenti, continue e differenziabili. Tuttavia, mentre nella cosine loss si perde la convessità e la loss di ListNet, come visto, presenta problemi legati al costo computazionale, la likelihood loss riesce a garantire la convessità pur rimanendo efficiente da calcolare. Pertanto, questi sono i motivi per cui si è deciso di adottare, all'interno dell'algoritmo ListMLE, il logaritmo della verosimiglianza.
Ciò significa che la funzione loss da ottimizzare nell'algoritmo ListMLE è data da

$$L(f; \mathbf{x}, \pi_y) = -\log P(\pi_y|\varphi(f(w, \mathbf{x}))) \tag{2.36}$$

Risulta evidente, quindi, il vantaggio computazionale che tale modifica comporta rispetto alla ListNet, in quanto in questo caso è sufficiente calcolare la probabilità della singola permutazione π_y, anziché quelle di tutte le $n!$ combinazioni possibili. Ancora una volta, è possibile utilizzare una rete neurale che sfrutti le tecniche di discesa del gradiente per ottimizzare la loss, ottenendo così il seguente algoritmo.

Algorithm 4 Algoritmo di addestramento di ListMLE

Require: dati del training set$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
  Parametri: learning rate η e tolleranza ϵ
  Inizializzare i pesi della rete neurale, componenti del vettore ω
  repeat
    for $i = 1, \ldots, m$ **do**
      Dai $x^{(i)}$, relativo alla query $q^{(i)}$ in input, alla rete neurale
      Calcola la lista degli score $z^{(i)}(f_\omega)$ utilizzando l' ω corrente.
      Calcola il gradiente $\Delta\omega$
      Aggiorna $\omega \leftarrow \omega - \eta\Delta\omega$
    end for
    Calcola la loss di verosimiglianza sul training set
  until il cambiamento della loss è minore di ϵ
  Output: i pesi della rete neurale, componenti del vettore ω

È necessario, però, sottolineare che, per come è stata presentata, la ListMLE accetta etichette reali solamente nel formato dell'intera lista corretta. Nel caso in cui esse vengano fornite come valori puntuali o binari, è possibile ottenere la permutazione reale adottando le tecniche di trasformazione spiegate nella sezione 2.3.
Nel 2014 è stata ideata una variante della ListMLE, definita *Position-Aware ListMLE* (o, in breve, p-ListMLE). Questo accorgimento è stato studiato in quanto erano emerse contraddizioni empiriche riguardanti la ListMle. Difatti, per come era implementato, l'algoritmo originale in alcune situazioni ignorava l'importanza della posizione, non rendendo tale procedura adeguata per risolvere opportunamente un problema di ranking, come si può vedere nell'esempio che segue.
Si consideri il caso in cui l'insieme di documenti è dato da $\mathbf{x}=\{x_1, \ldots, x_5\}$, la cui lista ideale prevede che le posizioni corrette siano date da $\mathbf{y}$=1,2,3,4,5. Si supponga, poi, che debbano essere valutate due funzioni di scoring f_1 e f_2 che attribuiscono, rispettivamente, i punteggi $y_1 = (\ln 4, \ln 5, \ln 3, \ln 2, \ln 1)$ e $y_2 = (\ln 5, \ln 4, \ln 1, \ln 2, \ln 3)$ secondo la legge $y_i(j) = f_i(x_j)$. Ciò significa che il primo predittore classifica erroneamente i primi due documenti, mentre il secondo sbaglia la posizione del terzultimo e dell'ultimo oggetto. Se si implementa l'algoritmo ListMLE utilizzando una funzione φ esponenziale, i valori di verosimiglianza logaritmica negativa che si ottengono nei due casi sono

$$L(f_1, \mathbf{x}, \mathbf{y}) = -\log\left(\frac{4}{4+5+3+2+1} \cdot \frac{5}{5+3+2+1} \cdot \frac{3}{3+2+1} \cdot \frac{2}{2+1} \cdot \frac{1}{1}\right)$$

$$L(f_2, \mathbf{x}, \mathbf{y}) = -\log\left(\frac{5}{5+4+3+2+1} \cdot \frac{4}{4+3+2+1} \cdot \frac{1}{1+2+3} \cdot \frac{2}{2+3} \cdot \frac{3}{3}\right)$$

Sviluppando i calcoli si ottiene che $L(f_1, \mathbf{x}, \mathbf{y}) < L(f_2, \mathbf{x}, \mathbf{y})$, pertanto f_1 risulta migliore di f_2 in termini della likelihood loss della ListMLE. Tuttavia, valutando l'operato dell'algoritmo secondo la metrica di valutazione NDCG@5, che si sa essere in grado di tenere

conto dell'importanza delle posizioni, si ottiene

$$\begin{aligned}\text{NDCG@5}(f_1, \mathbf{x}, \mathbf{y}) &= \frac{1}{N_5}\left((2^5-1)\log\frac{1}{3} + (2^4-1)\log\frac{1}{2} + (2^3-1)\log\frac{1}{4}\right) + \\ &+ \frac{1}{N_5}\left((2^2-1)\log\frac{1}{5} + (2^1-1)\log\frac{1}{6}\right) \\ \text{NDCG@5}(f_2, \mathbf{x}, \mathbf{y}) &= \frac{1}{N_5}\left((2^5-1)\log\frac{1}{2} + (2^4-1)\log\frac{1}{3} + (2^3-1)\log\frac{1}{6}\right) + \\ &+ \frac{1}{N_5}\left((2^2-1)\log\frac{1}{5} + (2^1-1)\log\frac{1}{4}\right)\end{aligned}$$

In questo caso, però, i calcoli mostrano che NDCG(f_1, $\mathbf{x}$, $\mathbf{y}$) < NDCG(f_2, $\mathbf{x}$, $\mathbf{y}$): giustamente la NDCG predilige il predittore f_2, poiché commette errori nelle posizioni meno rilevanti. Tuttavia, dato che la loss della ListMLE forniva il risultato opposto, la conclusione è che quest'ultima non è in grado di considerare l'importanza delle posizioni.
La contraddizione, quindi, risiede nel fatto che questa incapacità pervade la ListMLE nonostante la probabilità che la definisce, avendo un'impronta *top-down*, sembra poter gestire tale caratteristica. In realtà, riscrivendo tale probabilità, è chiaro come ogni posizione sia posta allo stesso livello da ListMLE. Difatti, la probabilità delle permutazioni può essere riscritta come

$$\begin{aligned}P(\mathbf{y}|\mathbf{x}; f) &= P(\mathbf{y}^{-1}(1), \mathbf{y}^{-1}(2), ..., \mathbf{y}^{-1}(n)|\mathbf{x}; f) \\ &= P(\mathbf{y}^{-1}(1)|\mathbf{x}; f)\prod_{i=2}^{n} P(\mathbf{y}^{-1}(i)|\mathbf{x}, \mathbf{y}^{-1}(1), ..., \mathbf{y}^{-1}(i-1); f)\end{aligned} \tag{2.37}$$

A questo punto, sfruttando la regola della catena, che stabilisce che $P(A_1, \ldots, A_n) = P(A_{i1})P(A_{i2}|A_{i1})\ldots P(A_{in}|Ai1, \ldots, A_{i-1})$, dove $(i_1, \ldots, i_n)$ è una qualunque permutazione di $(1, \ldots, n)$, è chiaro che la probabilità (2.37) può essere scomposta in ognuno dei seguenti modi

$$\begin{aligned}P(\mathbf{y}|\mathbf{x}; f) &= P(\mathbf{y}^{-1}(1), \mathbf{y}^{-1}(2), ..., \mathbf{y}^{-1}(n)|\mathbf{x}; f) \\ &= P(\mathbf{y}^{-1}(i_1)|\mathbf{x}; f)\prod_{j=2}^{n} P(\mathbf{y}^{-1}(i_j)|\mathbf{x}, \mathbf{y}^{-1}(i_1), ..., \mathbf{y}^{-1}(i_{j-1}); f)\end{aligned} \tag{2.38}$$

È evidente, perciò, che posizioni differenti assumono tutte la stessa importanza per la ListMLE e questo è il motivo per cui è stata introdotta la variante p-ListMLE.
L'idea di p-ListMLE consiste nella selezione di quei modelli che, passo dopo passo, partendo dal primo oggetto, inseriscono il documento corretto nella posizione indicata, considerando fissati i documenti trattati durante le iterazioni precedenti. Il procedimento, quindi, riflette il seguente pseudocodice.
Il problema appena presentato, quindi, è un problema di ottimizzazione sequenziale multiobiettivo. Pertanto, per semplificarne la risoluzione, lo si può rendere a obiettivo singolo applicando la tecnica della *linear scalarization*, così che il problema da ottimizzare diventi

$$\min_{f \in F} \Phi(f),$$

$$\text{dove } \Phi(f) = -\sum_{i=2}^{n} \alpha(i) \log P(\mathbf{y}^{-1}(i)|\mathbf{x}, \mathbf{y}^{-1}(1), ..., \mathbf{y}^{-1}(i-1); f) - \alpha(1) \log P(\mathbf{y}^{-1}(1)|\mathbf{x}; f) \tag{2.39}$$

Algorithm 5 Algoritmo p-ListMLE

Require: lista di documenti $\mathbf{x}$ da ordinare ed etichetta reale $\mathbf{y}$ contenente, in ordine, gli indici dei documenti della predizione ideale

Passo 1: trovare i modelli che massimizzano la probabilità che il primo oggetto selezionato sia $\mathbf{y}^{-1}(1)$, quello corretto, cioè risolvere

$$\max_{f \in F} P(\mathbf{y}^{-1}(1)|\mathbf{x}; f)$$

for $i = 2, \ldots, n$ **do**

Passo i: si denota con S_{i-1} il sottoinsieme di funzioni di ranking che hanno raggiunto il massimo al passo i-1. Al passo i, dati i primi i-1 oggetti classificati, si massimizza la probabilità che l'oggetto alla posizione i della permutazione reale venga selezionato, cioè si calcola

$$\max_{f \in S_{i-1}} P(\mathbf{y}^{-1}(i)|\mathbf{x}, \mathbf{y}^{-1}(1), \ldots, \mathbf{y}^{-1}(i-1); f)$$

end for

Output: una funzione di ranking f scelta casualmente all'interno dell'insieme S_{n-1}

dove il primo termine è stato esplicitato per sottolineare che esso è ottenuto senza fissare nessun altro documento in precedenza e dove $\alpha(\cdot)$ è una funzione decrescente. In particolare, incorporando la probabilità (2.30) di Plackett-Luce, si ottiene che la loss di verosimiglianza dell'algoritmo p-ListMLE è data da

$$L_p(f; \mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \alpha(i) \left(-f(x_{\mathbf{y}^{-1}(i)}) + \log \left(\sum_{j=1}^{n} e^{f(x_{\mathbf{y}^{-1}(j)})} \right) \right) \tag{2.40}$$

Ciò significa che questa variante mantiene i vantaggi di ListMLE spiegati in precedenza e, in più, è in grado di tenere in considerazione l'importanza delle posizioni, motivo per cui si è scelto di utilizzarla come loss listwise all'interno del progetto.
Analizzando nuovamente l'esempio precedente, infatti, i valori delle loss p-ListMLE relativi alle funzioni f_1 e f_2 assumono valore

$$\begin{aligned} L_p(f_1, \mathbf{x}, \mathbf{y}) = &-\alpha(1) \log \left(\frac{4}{4+5+3+2+1} \right) - \alpha(2) \log \left(\frac{5}{5+3+2+1} \right) + \\ &- \alpha(3) \log \left(\frac{3}{3+2+1} \right) - \alpha(4) \log \left(\frac{2}{2+1} \right) \end{aligned}$$

$$\begin{aligned} L_p(f_2, \mathbf{x}, \mathbf{y}) = &-\alpha(1) \log \left(\frac{5}{5+4+3+2+1} \right) - \alpha(2) \log \left(\frac{4}{4+3+2+1} \right) + \\ &- \alpha(3) \log \left(\frac{1}{1+2+3} \right) - \alpha(4) \log \left(\frac{2}{2+3} \right) \end{aligned}$$

Perciò, si ha consistenza con la NDCG se vale che

$$\begin{aligned} L_p(f_1, \mathbf{x}, \mathbf{y}) - L_p(f_2, \mathbf{x}, \mathbf{y}) &= (\alpha(1) - \alpha(2) - \alpha(4)) \ln 5 - (\alpha(1) - \alpha(2)) \ln 4 + \\ &\quad - (\alpha(3) - \alpha(4)) \ln 3 + \alpha(2)(\ln 11 - \ln 10) \\ &> (\alpha(1) - \alpha(2) - \alpha(4)) \ln 5 - (\alpha(1) - \alpha(2) + \alpha(3) - \alpha(4)) \ln 4 \end{aligned}$$

la quale è soddisfatta solo se

$$\frac{\alpha(1) - \alpha(2) - \alpha(4)}{\alpha(1) - \alpha(2) + \alpha(3) - \alpha(4)} > \frac{\ln 4}{\ln 5}$$

Ma perché essa sia verificata è sufficiente che $\alpha(1)$ sia abbastanza maggiore di $\alpha(i)$, per $i = 2, \ldots, 5$. Poiché questa è una condizione facile da soddisfare, la p-ListMLE è consistente con la NDCG, ovvero il risultato che si cercava di ottenere.
In generale, si è dimostrato che buone performance si ottengono quando α è strutturato come il fattore di guadagno della NDCG, utilizzando come etichetta all'esponente la sottrazione tra il totale dei documenti e l'indice di posizionamento della lista ottimale così che α risulti crescente, cioè scegliendo $\alpha(i) = 2^{n-y_i} - 1$.

2.6.5 Valutazione dell'approccio listwise

L'analisi effettuata sugli algoritmi listwise consente di poter confermare le sensazioni suggerite inizialmente, ossia che questi ultimi metodi risolvono il problema del ranking con un approccio migliore rispetto agli algoritmi pointwise e pairwise. Difatti, visionando una query per volta nella sua globalità, questi metodi sono in grado di distinguere quali documenti appartengono alla stessa interrogazione, proprietà che, come è stato dimostrato, non è verificata negli altri due approcci. Inoltre, l'approccio listwise, teoricamente, è sviluppato appositamente per tenere in considerazione l'informazione relativa alla posizione dei documenti all'interno della classifica. Esistono, infatti, studi che confermano questo teorico incremento nelle performance anche su esperimenti pratici, risultato che sarà confermato dal progetto realizzato.
Tuttavia, sono presenti anche aspetti negativi collegati agli approcci listwise. Innanzitutto, come si è potuto osservare in ListNet, se le funzioni loss non vengono scelte opportunamente, i costi computazionali necessari per l'implementazione possono crescere in modo estremamente rapido. In aggiunta a ciò, nonostante possa sembrare il contrario, non sempre gli algoritmi listwise tengono effettivamente conto della posizione dei documenti. Si è visto, infatti, che l'algoritmo ListMLE, non essendo costruito mediante un esplicito fattore di sconto della posizione, è dovuto essere stato modificato per far sì che godesse di tale proprietà. Tutto ciò è dovuto alla prossimità di tali metodi, che può essere sia uno svantaggio che un punto di forza.
Da un certo punto di vista, infatti, le nuove strutture su cui questo approccio si basa non consentono un'implementazione diretta tramite modelli noti e ben studiati, che invece si sono potuti sfruttare nei due approcci precedenti. Pertanto, è necessario effettuare anche approfonditi studi teorici che garantiscano convergenza e implementabilità di tali algoritmi.
D'altro canto, il fatto che il mondo dei metodi listwise sia ancora lontano dall'essere completamente esplorato, rende possibile la scoperta di inedite strategie e nuovi accorgimenti in grado di alzare ancora di più l'asticella dello stato dell'arte raggiunto da questi metodi. Un esempio, in questo senso, è dato dallo studio recente della proprietà dell'*attention*: sfruttando delle reti neurali ricorrenti si sta cercando, infatti, di costruire, a partire dai documenti iniziali, nuove rappresentazioni basate sull'importanza relativa di un documento all'interno della query. Per ottimizzare i tempi, poiché le posizioni importanti sono le prime, è meglio concentrarsi sui documenti più importanti. Ma l'importanza dipende da quanti altri altri documenti rilevanti sono presenti all'interno della query: assume quindi

maggior interesse un documento mediamente rilevante all'interno di una query costituita solamente da esempi irrilevanti, piuttosto che un oggetto rilevante ma di cui si hanno già altre migliaia di esempi significativi relativi alla stessa interrogazione. Sfruttando poi la ricorrenza della rete, questa rappresentazione viene mandata indietro alla rete stessa così che si possa addestrare il modello utilizzando questo nuovo dataset.
Infine, per dimostrare la vicinanza temporale di questi metodi e lo studio che ancora li riguarda, è importante sottolineare che, nel 2018, si è dimostrato che tutte le loss dei metodi listwise finora sviluppati possono essere considerate un caso particolare di una generica formulazione. Questa loss generale, che è una funzione costo di negativa verosimiglianza logaritmica, prende il nome di *LambdaLoss* ed è definita come

$$L(\mathrm{s}, \mathrm{y}) = -\log P(\mathrm{y}|\mathrm{s}) = -\log \sum_{\pi \in \Omega} P(\mathrm{y}|\mathrm{s}, \pi) P(\pi|\mathrm{s}) \tag{2.41}$$

dove il primo fattore della sommatoria è il termine di verosimiglianza e la seconda probabilità rappresenta la distribuzione della lista predetta. Una tale generalizzazione è molto importante, perché per il futuro permetterà di ottenere fondamenta generiche più precise e di ricavare un maggior numero di loss studiando i casi particolari che la (2.41) propone.

2.7 Analisi della relazione Metodo-NDCG

In quest'ultimo approfondimento teorico verrà analizzata la relazione che intercorre tra i vari approcci presentati e le metriche di valutazione, in particolare la NDCG. Oltre alla preparazione del dataset, infatti, si è visto che la componente che più differenzia i tre metodi è la funzione loss attraverso cui la macchina apprende. Quindi, lo scopo di questa sezione riguarderà proprio l'analisi del legame tra funzioni di costo e NDCG@m, cioè la NDCG calcolata su tutti gli m documenti della query, che, d'ora in avanti, per semplicità, sarà indicata solamente con NDCG. In particolare, supponendo che le informazioni reali vengano passate attraverso la permutazione corretta, la NDCG calcolata rispetto ad una lista π assume la forma

$$\mathrm{NDCG}(\pi, \Omega_y) = \frac{1}{Z_m} \sum_{t=1}^{m} G(\pi_y^{-1}(t)) \eta(\pi(\pi_y^{-1}(t))), \quad \forall \pi_y \in \Omega_y \tag{2.42}$$

dove Z_m è il valore massimo raggiunto dalla DCG e η rappresenta il fattore di sconto della NDCG. Talvolta si specificherà anche $\mathrm{NDCG}(f, \mathbf{x}, \Omega_y)$, nel caso in cui si voglia sottolineare che l'ordinamento finale sia stato ottenuto dalla composizione di una funzione di ordinamento con la funzione di scoring f. L'obiettivo, quindi, consisterà nella ricerca di un upper bound, dipendente dalla loss studiata, della quantità NDCG, così da investigare se, minimizzando la loss, diminuisce il valore 1-NDCG e, di conseguenza, aumenta la NDCG.
Per quanto riguarda gli algoritmi pointwise, Cossock e Zhang [14] hanno dimostrato che tale limite, in termini di una loss di regressione, esiste e può essere scritto come

$$1 - \mathrm{NDCG}(f, \mathbf{x}, \Omega_y) \leq \frac{1}{Z_m} \left(2 \sum_{j=1}^{m} \eta_j^2 \right)^{\frac{1}{2}} \left(2 \sum_{j=1}^{m} (f(x_j) - y_j)^2 \right)^{\frac{1}{2}} \tag{2.43}$$

Con le stesse tecniche, i due autori hanno derivato un'espressione simile anche nel caso di funzioni loss di classificazione.
Analizzando l'equazione (2.42) è possibile osservare che la minimizzazione della loss comporta una massimizzazione della NDCG e questa è un'importantissima proprietà positiva. Tuttavia, può accadere anche che il primo termine, cioè 1-NDCG, sia 0 nonostante il membro di destra, la funzione loss, sia lontano dall'essere nullo. Ciò significa che la minimizzazione di tale loss è soltanto una condizione sufficiente affinché la NDCG sia massima, non anche necessaria. E questo è ovvio, dal momento che nel termine di sinistra è presente esplicitamente un fattore di sconto della posizione che, al contrario, a destra risulta assente. Tale comportamento, infatti, era già stato notato nella sezione 2.2 relativa alle metriche, dove un esempio mostrava la possibilità di ottenere una classificazione perfetta nonostante predizioni di singoli punteggi totalmente sbagliate. Anche questa analisi, quindi, riflette l'incapacità, già evidenziata, dei metodi pointwise di trattare adeguatamente il problema del ranking.
Per quanto riguarda i metodi pairwise e listwise, invece, all'interno dello stesso articolo si è adottata una strategia comune per dimostrare che le relative funzioni loss limitano la quantità 1-NDCG. Per ottenere questo risultato è stata introdotta una nuova, generica, loss, detta *funzione loss essenziale*. L'idea è dimostrare che tale funzione sia un limite superiore di 1-NDCG e che, al tempo stesso, le funzioni loss degli algoritmi studiati siano una soglia della funzione loss essenziale. Se questo accade, quindi, le singole funzioni loss saranno, a loro volta, estremi superiori di 1-NDCG. Innanzitutto è necessario definire la funzione loss essenziale.
Dato un gruppo di documenti $\mathbf{x}$ e la loro permutazione reale $\pi_y \in \Omega_y$, l'idea è scomporre il problema del ranking in una serie di sotto-problemi iterativi. Ad ogni passo t, quindi, si deve individuare, tra tutti i documenti classificati al pari o al di sotto della posizione t-esima nella permutazione π_y, quello posizionato esattamente come t-esimo, ossia $\pi_y^{-1}(t)$. Per raggiungere questo obiettivo si utilizza una funzione di scoring f. Basandosi su f, allora, si può costruire il predittore

$$T \circ f(\mathbf{x}_{(t)}) = \arg \max_{j=\pi_y^{-1}(t),\ldots,\pi_y^{-1}(m)} f(x_j), \quad T = \text{sort} \tag{2.44}$$

il cui output sarà proprio $\pi_y^{-1}(t)$. È ovvio, quindi, che gli output sono $m - t$, poiché possono essere $\{\pi_y^{-1}(t), \ldots, \pi_y^{-1}(m)\}$. Pertanto, la loss 0-1, quella che conta gli errori, in questo caso può essere scritta come

$$\begin{aligned} L_t(f; \mathbf{x}(t), \pi_y^{-1}(t)) &= I_{\{T \circ f(\mathbf{x}_{(t)}) \neq \pi_y^{-1}(t)\}} \\ &= 1 - \prod_{j=t+1}^{m} I_{\{f(\pi_y^{-1}(t)) > f(\pi_y^{-1}(j))\}} \end{aligned} \tag{2.45}$$

A questo punto, sommando le loss di ogni passo $t = 1, \ldots, m-1$, si ha

$$\tilde{L}(f; \mathbf{x}(t), \pi_y) = \sum_{t=1}^{m-1} I_{\{T \circ f(\mathbf{x}_{(t)}) \neq \pi_y^{-1}(t)\}} \tag{2.46}$$

Infine, minimizzando su tutto l'insieme di permutazioni Ω_y, si ottiene la funzione loss essenziale

$$\tilde{L}(f; \mathbf{x}(t), \Omega_y) = \min_{\pi_y \in \Omega_y} \sum_{t=1}^{m-1} I_{\{T \circ f(\mathbf{x}_{(t)}) \neq \pi_y^{-1}(t)\}} \tag{2.47}$$

Gli stessi autori, poi, hanno dimostrato che tale funzione è effettivamente un limite superiore di 1-NDCG, in quanto vale che

$$1 - NDCG(f, \mathbf{x}(t), \Omega_y) \leq \frac{2^{K-1} - 1}{Z_m} \left(\sum_{t=1}^{m-1} \eta(t)^{\alpha} \right)^{\frac{1}{\alpha}} \left(\tilde{L}(f; \mathbf{x}(t), \Omega_y) \right)^{\frac{1}{\beta}} \tag{2.48}$$

dove $\frac{1}{\alpha} + \frac{1}{\beta} = 1$. Successivamente, sfruttando la proprietà della loss essenziale secondo cui $\tilde{L}(f; \mathbf{x}(t), \Omega_y) \leq \max_{\pi_y \in \Omega_y} a(T \circ f(\mathbf{x}_{(t)}), \pi_y^{-1}(t))$, dove a è la funzione loss 0-1, si può dimostrare che le funzioni loss pairwise limitano dall'alto la loss essenziale, in quanto vale che

$$\tilde{L}(f; \mathbf{x}(t), \Omega_y) \leq \sum_{t=1}^{m-1} \sum_{\substack{j=t+1 \\ l_{\pi_y^{-1}(t)} \neq l_{\pi_y^{-1}(j)}}}^{n} \varphi(f(x_{\pi_y^{-1}(t)}) - f(x_{\pi_y^{-1}(j)})), \quad \forall \pi_y \in \Omega_y \tag{2.49}$$

Infatti, se al posto della φ si sostituiscono le funzioni loss esponenziali e logistica, si ottengono, rispettivamente, le funzioni loss di RankBoost e di RankNet.
Rispetto alla limitazione trovata per i metodi pointwise, la (2.48) implica che, quando la quantità 1-NDCG è pari a 0, anche il termine di destra debba essere 0. Perciò, in questo caso, l'annullamento della loss è una condizione sia necessaria che sufficiente per la massimizzazione della NDCG, eventualità che individua una proprietà positiva.
Per quanto riguarda le loss listwise non vengono riportati le specifiche limitazioni, in quanto i risultati ottenuti ed i procedimenti utilizzati sono analoghi a quelli appena descritti riguardanti le loss pairwise. In particolare, quindi, la correlazione tra loss e 1-NDCG è valida sia per la ListMLE che per la ListNet. Per quest'ultima, nello specifico, si è potuto dimostrare che tale considerazione rimane valida anche considerando la 1-NDCG@5.
Tuttavia, va considerato che questi studi non possono essere sufficienti come giustificazione teorica. Difatti, le proprietà appena dimostrate sono certamente positive, ma non è detto che il punto ottimale della loss, cioè dell'estremo superiore, coincida con il valore che rende ottima la quantità 1-NDCG. Questo è il motivo per cui, tra gli approcci listwise, si sono sviluppate anche tecniche in grado di ottimizzare direttamente le metriche di valutazione tipo la NDCG. Come già spiegato nella sezione precedente, quindi, è necessario approfondire gli studi teorici in grado di giustificare l'utilizzo di questi metodi innovativi, in particolare andando ad investigare il rapporto tra le funzioni loss e la consistenza spiegata nella definizione 1.4, poiché questa grandezza è quella che descrive, nella maniera più accurata possibile, il legame tra la funzione costo utilizzata e la grandezza che si vuole effettivamente minimizzare.

Capitolo 3

Applicazioni del Learning to Rank

Nei capitoli precedenti ci si è soffermati sulle proprietà teoriche degli algoritmi inerenti al learning to rank, fornendo una primi analisi riguardante i vantaggi e gli svantaggi di ciascun approccio. Tuttavia, non è ancora stato mostrato un esempio concreto di questi studi. I prossimi argomenti trattati si concentreranno proprio su questo aspetto.
In particolare, verranno innanzitutto considerati alcuni utilizzi concreti dei precedenti algoritmi. Nella sezione 3.1 saranno presentate diverse applicazioni del Learning to Rank, per le quali verranno descritti e confrontati i risultati della sperimentazione numerica presenti in letteratura. La sezione 3.2 è dedicata all'analisi dettagliata di un problema di ranking di documenti, per il quale i diversi approcci sono stati implementati in ambiente Python e confrontati tra loro. Sarà così possibile cogliere le innumerevoli applicazioni di questi modelli e, di conseguenza, l'importanza dell'approfondimento del loro studio. Infine, si procederà con l'illustrazione di un progetto, realizzato in prima persona nell'ambito di un tirocinio aziendale, relativo ad un problema di ranking di documenti.
A tal riguardo, prima verrà spiegato il procedimento con cui si è deciso di affrontare il problema specifico e, in seguito, verrà effettuata un'indagine sui risultati ottenuti volta a capire se le conclusioni teoriche raggiunte in precedenza siano confermate o meno dagli esperimenti pratici.

3.1 Esempi concreti di utilizzo del Learning to Rank

3.1.1 Sentiment Analysis

In precedenza è già stato spiegato che il learning to rank e, in generale, l'information retrieval possono essere applicati in numerosissimi ambiti. Tra questi, la sentiment analysis assume un ruolo centrale, in quanto risponde a svariate esigenze. Con il termine *sentiment analysis* si intendono i processi che consentono ad una macchina di interpretare frasi umane, in modo da cogliere il giudizio e le opinioni che l'interlocutore ha rispetto al tema trattato.
Il legame con il learning to rank, quindi, può essere trovato in svariate applicazioni. Infatti, avendo la possibilità di capire le preferenze dell'utente, si possono costruire sistemi di raccomandazione capaci, a partire da ciascun profilo, di stilare una classifica degli oggetti a lui più inerenti. Esempi specifici di tale applicazione possono essere gli shop online come Amazon, che consigliano gli oggetti più apprezzati tra tutti gli articoli disponibili nel catalogo. Il learning to rank è perfetto per svolgere questo compito, in quanto è neces-

sario stilare una classifica di preferenza e similarità rispetto agli acquisti e, soprattutto, le recensioni precedentemente realizzate. Ovviamente, come già accennato nel secondo capitolo, Amazon non è l'unico esempio esistente, ma anche siti come Booking e Netflix sfruttano queste tecniche per suggerire i prodotti preferiti dal potenziale acquirente.
In particolare, per quanto riguarda la raccomandazione dei film, esiste un dataset di riferimento su cui poter effettuare esperimenti per capire la tecnica di learning to rank più adatta da utilizzare. Il dataset è chiamato *MovieLens 100K dataset* ed è stato creato dai ricercatori del *GroupLens Research Project* dell'Università del Minnesota. È costituito da 100.000 valutazioni raccolte dal sito movielens.umn.edu (con punteggi da 1 a 5) effettuate ad opera di 943 utenti, tra il 1997 ed il 1998, riguardo a 1682 film. Inoltre, per ciascun utente, che deve aver recensito almeno 20 film, sono presenti anche alcune informazioni personali, come l'età ed il sesso. Non essendo molto vasto, il dataset appena descritto non è considerato come il miglior modello su cui effettuare comparazioni di prestazioni di algoritmi di learning to rank, ciononostante è stato comunque utilizzato in svariati casi. Infatti, all'interno delle guide di *TensorFlow*, una delle più famose librerie software per l'apprendimento automatico, sono presenti esempi basati proprio sul dataset Movielens 100K. In particolare, in [1] è presentata un'implementazione dell'algoritmo ListNet i cui risultati sono valutati attraverso la metrica NDCG. Inoltre, ancora una volta in [1] è proposta un'analisi di comparazione tra algoritmi pointwise, pairwise e listwise, implementati, rispettivamente, con la loss Mean Squared Error, la Hinge loss e la p-ListMLE. Anche in questo caso la valutazione viene effettuata tramite la NDCG e le conclusioni mostrano che, nonostante in generale i risultati siano ottimi, l'algoritmo che meglio performa è quello listwise, come peraltro suggerisce la teoria.

3.1.2 Ohsumed, il dataset medico

Un altro dataset molto famoso e utilizzato spesso come riferimento è l'*Ohsumed*. Esso fa parte di una collezione di dataset denominata *LETOR*, da LEarning TO Rank, creata dalla Microsoft Reserach Asia, la cui prima versione risale al 2007 e che ora è aggiornata fino alla quarta, completata nel 2009. La creazione di questa raccolta di dataset specifici per gli obiettivi di learning to rank ha consentito agli studi in questa disciplina di affrontare un repentino miglioramento.
Nei problemi di machine learning è pratica comune utilizzare dataset di riferimento per testare l'efficacia di nuovi algoritmi studiati solo da un punto di vista teorico. Difatti, tale procedura comporta due vantaggi principali: in primo luogo permette ai ricercatori di impiegare il tempo a disposizione esclusivamente nell'implementazione del modello, evitando la raccolta e la pulizia dei dati, che spesso è la parte più dispendiosa dello studio; in aggiunta, l'utilizzo di un dataset comune, uguale per tutti gli esperimenti, consente di effettuare comparazioni più efficaci tra un algoritmo e l'altro, poiché i dati su cui essi vengono addestrati e testati sono i medesimi. La creazione di LETOR, quindi, ha permesso che ciò fosse possibile anche per il learning to rank.
All'interno di questa raccolta sono presenti due gruppi principali di dataset. Il primo, detto *Gov Corpus*, è costituito da pagine web ed è utile negli esperimenti riguardanti i motori di ricerca. In questo ambito, poi, sono stati creati altri dataset di riferimento, come il *MSLR-WEB10K*, realizzato sempre dalla Microsoft e su cui verrà svolto il progetto che sarà discusso in seguito. La seconda raccolta è proprio l'Ohsumed. Questo dataset è interessante perché permette di illustrare un altro campo in cui il learning to rank si rivela

essere utile, quello della medicina. Questa raccolta è un sottoinsieme di *MEDLINE*, un database contenente pubblicazioni mediche. In tutto, all'interno di Ohsumed, sono presenti 16.140 coppie del tipo documento-query relative a 106 differenti query. Ogni documento ha una rappresentazione vettoriale ed è descritto da 45 features, tra cui anche i valori di BM25 e di TF, mentre le etichette possibili sono tre e si suddividono in *decisamente rilevante*, *possibilmente rilevante* ed *irrilevante*, ognuna delle quali è stata associata ad una coppia da un essere umano. Ciascuna query rappresenta una ricerca medica operata da un dottore, pertanto, in risposta, la macchina dovrà restituire quegli articoli scientifici che, a seconda delle condizioni del paziente e della malattia studiata, siano in grado di fornire risposte in merito alle cure e ai trattamenti da eseguire. Essendo uno dei dataset più utilizzati, anche in questo caso si possono trovare risultati che confrontano tutti e tre gli approcci del learning to rank. In particolare, in [47] è possibile osservare le implementazioni di vari metodi ed un loro confronto ottenuto mediante il calcolo della NDCG limitata a differenti posizioni. Ancora una volta, è possibile notare come gli algoritmi più efficaci siano quelli listwise, in particolare ListMLE.

3.1.3 Lo sport, dalle discipline individuali a quello di squadra

Oltre agli esempi relativi ai dataset di riferimento già pre-costruiti, se si riescono ad accumulare abbastanza informazioni affinché il modello sia in grado di individuare schemi da cui imparare a generalizzare, le applicazioni in cui il learning to rank è in grado di ottenere importanti risultati sono molteplici. Ovviamente le difficoltà in questi casi aumentano in maniera notevole, in quanto, dopo aver accumulato i dati, è necessario capire come e quali features estrarre per rappresentare significativamente tali dati. In particolare, in questo paragrafo l'attenzione sarà rivolta al mondo dello sport, mostrando uno studio riguardante una disciplina individuale, il ciclismo, ed un possibile approccio adatto agli sport di squadra focalizzato sulla pallacanestro.
Per quanto riguarda il ciclismo, lo studio [21] è stato effettuato dal Dipartimento di computer science e di matematica dell'Università belga di Antwerp. L'idea è stata quella di associare ogni corsa in linea - corsa in cui i corridori partono contemporaneamente e la classifica viene stilata in base all'ordine di arrivo, senza la presenza di tappe - ad una query i cui relativi documenti da ordinare sono i corridori. In questo contesto diventa evidente la possibilità di utilizzo delle tecniche di learning to rank per "ordinare" la lista d'arrivo dei corridori e prevedere, in questo modo, il vincitore della corsa. Già in passato, infatti, era stato sfruttato il machine learning anche in questo ambito, ad esempio per prevedere l'esatto tempo impiegato da ciascun ciclista per completare la gara. Tuttavia, secondo lo stesso ragionamento teorico che predilige un metodo listwise ad uno pointwise, la possibilità di servirsi del learning to rank per prevedere esclusivamente l'ordine di arrivo, senza doversi concentrare su un aspetto intermedio molto più difficile da ottenere con certezza, come appunto il tempo di percorrenza, ha spinto gli autori a investigare questa strada.
Innanzitutto, lo studio si è soffermato sulla comprensione dei dettagli in grado di influire maggiormente su una corsa. Nonostante un algoritmo di predizione assolutamente perfetto sia impossibile da costruire, in quanto gli eventi sportivi sono condizionati da fattori esterni e imprevedibili, come il meteo o le cadute degli atleti, rappresentare i dati in un maniera più o meno significativa può comportare una grande differenza di prestazione.
Le features utilizzate riguardano i dati storici delle gare analizzate, le singole performance

degli ultimi anni degli atleti e i loro dati anagrafici. In questo modo è possibile capire che caratteristiche debba avere un corridore per primeggiare in una certa corsa e quale sia l'andamento sportivo che riguarda ogni atleta. Dopo aver costruito le features ritenute più opportune, è stato adottato come algoritmo di learning to rank *LambdaMART*, un adattamento ad una struttura boosting di alberi di LambdaRank, l'algoritmo analizzato nella sezione 2.5.4. Addestrando il modello sui dati delle stagioni comprese tra il 2000 ed il 2017, si sono effettuate predizioni sulle corse del 2018, 2019 e 2021 (il 2020 è stato escluso a causa della forte imprevedibilità dovuta alla pandemia di COVID-19). Utilizzando come metrica di valutazione la NDCG@10 (in quanto, come nell'information retrieval, anche in questo ambito le posizioni più importanti sono le prime), si sono rapportati i risultati ottenuti alle medie delle predizioni realizzate da fan ed esperti. Il risultato è stato che il modello addestrato tramite il learning to rank si è dimostrato capace di migliorare tali previsioni, ottenendo un valore medio di NDCG@10 pari a 0.55, con punte anche superiori a 0.8. Ovviamente questo modello è ancora lontano dalla perfezione e per certi aspetti, riguardanti soprattutto giovani atleti emergenti, per i quali mancano informazioni pregresse, risulta essere peggiore delle valutazioni umane. Tuttavia, in generale, questo approccio ha mostrato grandi potenzialità e ciò permette di aiutare gli allenatori, ed i corridori stessi, a capire quali avversari possano rappresentare l'ostacolo maggiore e quale strategia sia più efficace adottare in una certa corsa, ma anche i giornalisti sportivi ed i tifosi (nonché gli appassionati di scommesse) per suggerire cosa quali risultati ci si possano aspettare in un certo evento.

Per quanto riguarda gli sport di squadra, invece, è possibile adottare una strategia differente, come nel caso illustrato in [39] che si concentra sul mondo della pallacanestro. Infatti, per rappresentare gli incontri che avvengono tra le squadre, si utilizza una rappresentazione a grafo dove i nodi rappresentano le franchigie e gli archi che collegano due entità contengono le informazioni relative all'incontro rappresentato. Poiché PageRank è in grado di gestire il flusso delle informazioni all'interno di un grafo, la strategia proposta prevede di adattare tale tecnica per fare fluire la "forza" di un nodo da una squadra all'altra, a seconda dei risultati delle partite. Questo avviene tramite una procedura iterativa che considera contemporaneamente vari aspetti, al fine di ottenere la legge corretta che consenta di rappresentare nel migliore dei modi la situazione globale delle forze del campionato. In particolare, nel caso dello studio riportato, queste caratteristiche prevedono sia peculiarità del basket, come i punti, gli assist ed i rimbalzi conquistati - che avranno un'incidenza positiva - ed i falli commessi ed i possessi persi - che invece influiranno negativamente sulla forza di una squadra - sia proprietà generiche, caratterizzate, ad esempio, dalla forza della squadra avversaria e dal luogo dell'incontro effettuato, ossia se in casa o in trasferta. Dopo essersi procurati un quadro accurato della situazione, è stata implementata una tecnica di learning to rank specifica per il caso studiato, creando una loss basata sull'estensione del modello di PageRank illustrato. In questo modo è stato possibile prevedere, stagione per stagione, dopo aver analizzato i dati della prima metà del campionato, quali squadre avrebbero raggiunto i play-off e quale avrebbe poi vinto il campionato, ottenendo anche in questo caso risultati soddisfacenti.

3.1.4 Costruzione di un portfolio finanziario basato sulla tecnica del long-short

Un'ulteriore applicazione del learning to rank si può trovare nel mondo della finanza. Recentemente, infatti, l'utilizzo dell'intelligenza artificiale sta prendendo sempre più piede nel suggerire agli investitori su quali titoli investire o meno. Ancora una volta, quindi, il learning to rank può adattarsi perfettamente per rispondere a questa esigenza. In particolare, come è spiegato nell'articolo [49] del 2020, esso è in grado di suggerire una strategia long-short ottimale.
In ambito finanziario, una strategia long-short per la costruzione di un portfolio è una tecnica di investimento che cerca di sfruttare, contemporaneamente, sia i titoli promettenti che quelli manifestanti un andamento a ribasso. L'idea, infatti, consiste, in una prima fase, nell'ordinare i titoli dal più al meno promettente. Successivamente, sulle azioni in crescita si va "lungo", che significa effettuare una classica compravendita in cui il guadagno deriva dall'acquisto di un titolo ad un certo prezzo ed una sua futura rivendita ad un costo superiore. Al tempo stesso, però, sui titoli caratterizzati da un andamento peggiore si va "corto": si effettua una vendita allo scoperto in cui prima si decide di vendere un'azione che ancora non si possiede al prezzo attuale, impegnandosi poi ad acquistarla successivamente. Effettuando questa operazione su un titolo il cui valore sta diminuendo è possibile incrementare il proprio guadagno, in quanto l'acquisto futuro avverrà ad un prezzo inferiore rispetto a quello concordato per la vendita.
La ricerca di un corretto ordinamento dei titoli suggerisce che il learning to rank è adatto ad essere utilizzato in questo ambito. Tuttavia, rispetto alla classica formulazione del ranking di documenti, in questo caso sorgono dei problemi aggiuntivi. Innanzitutto, i dati trattati non riguardano più singoli documenti immutabili, ma sono costituiti da serie temporali, che richiedono maggiori attenzioni. Ma, soprattutto, ciò che più si differenzia dalle strategie di learning to rank finora proposte risiede nel fatto che i documenti classificati negli ultimi posti non possono più essere trascurati: al contrario, l'attenzione ora deve risultare massima anche per le posizioni finali dell'ordinamento, così da capire su quali azioni sia meglio effettuare operazioni di short. Uno scenario completamente differente da quello proposto finora.
Una delle prime strategie proposte per adattarsi a questo contesto prevede di effettuare l'addestramento del modello e la successiva classificazione dello stesso dataset due volte, invertendo, nel secondo caso, le label. Questa procedura però, oltre ad essere più costosa sia in termini di tempo di addestramento che di impegno nel settaggio dei parametri, può suggerire di vendere ed acquistare la stessa azione, causando contraddizioni irrisolvibili.
È per questo motivo, quindi, che nell'articolo sopracitato viene proposta una variante della ListMLE, chiamata dagli autori *ListFold*, in grado di porre allo stesso livello di importanza sia le prime che le ultime posizioni dell'ordinamento. In particolare, l'intuizione da cui è stata sviluppata ListFold prevede di adottare nuovamente il modello Plackett-Luce, ma con un'implementazione differente: nel momento in cui la probabilità della permutazione è scomposta nel prodotto di probabilità computazionali, queste ultime riguardano non più un singolo oggetto ma analizzano le coppie, così da restituire, ad ogni passo, i due elementi caratterizzati dalla maggior differenza di punteggio. In questo modo, ad ogni iterazione, si determina una coppia long-short.
Quindi, supponendo di dover classificare un numero pari di oggetti, denotati con $X_1, \ldots, X_{2n}$,

la formula (2.30) diventa

$$P(\pi|\mathbf{s}) = \prod_{j=1}^{n} \frac{\varphi(s_{\pi^{-1}(j)} - s_{\pi^{-1}(2n+1-j)})}{\sum_{j \le u \ne v \le 2n+1-j} \varphi(s_{\pi^{-1}(u)} - s_{\pi^{-1}(v)})} \tag{3.1}$$

Di conseguenza, la funzione loss di negativa verosimiglianza logaritmica è data da

$$\begin{aligned} L(f; \mathbf{x}, \pi_y) &= -\log P(\pi|\mathbf{s}) \\ &= -\sum_{i=1}^{n} \left(\log \varphi(s_{\pi^{-1}(j)} - s_{\pi^{-1}(2n+1-j)}) - \log \sum_{j \le u \ne v \le 2n+1-j} \varphi(s_{\pi^{-1}(u)} - s_{\pi^{-1}(v)}) \right) \end{aligned} \tag{3.2}$$

Dato che nella situazione appena presentata l'obiettivo è differente, occorre che anche le metriche di valutazione siano appropriatamente aggiornate. In particolare, la NDCG verrà generalizzata alla NDCG@±k, una nuova metrica in grado di valutare la classificazione sia delle prime che delle ultime posizioni e definita da

$$\begin{aligned} \text{NDCG@} \pm k(\pi, l) &= (\text{NDCG@}k(\pi, l) + \text{NDCG@} - k(\pi, \tilde{l}))/2 \\ &= \frac{1}{2Z_k} \left(\sum_{j=1}^{k} G(l_{\pi^{-1}(j)}) \eta(j) + \sum_{j=1}^{k} G(\tilde{l}_{\tilde{\pi}^{-1}(j)}) \eta(j) \right) \end{aligned} \tag{3.3}$$

dove $\tilde{\pi}$ e $\tilde{l}$ indicano, rispettivamente, la permutazione e le label inverse.
Valutando l'algoritmo ListFold su un dataset costituito da azioni del mercato cinese, si può vedere come i risultati presentati nell'articolo confermino che l'utilizzo di una procedura di learning to rank adeguatamente implementata comporta il raggiungimento dei migliori risultati possibili.

3.2 MSLR-WEB1OK: un problema di ranking di documenti

Tra le varie applicazioni che già erano state citate nei capitoli precedenti, ricorrente era l'esempio del motore di ricerca. In questo caso, a partire dai documenti forniti, il modello deve essere capace di restituire gli esempi più rilevanti in risposta ad una precisa richiesta dell'utente. Poiché questo problema è comune negli studi di learning to rank, è stato scelto di affrontare nel dettaglio proprio questa applicazione per comprendere appieno e toccare con mano le implicazioni che derivano dalla teoria analizzata. In particolare, il dataset utilizzato sarà il *MSLR-WEB10K*. In questo paragrafo, quindi, verrà approfondito il progetto svolto, che prevede l'implementazione di differenti approcci di learning to rank ed il relativo confronto in merito ai risultati ottenuti. Inizialmente, quindi, verranno illustrati il dataset utilizzato e le operazioni di pre-processing attuate prima di memorizzarlo. In seguito, saranno spiegati gli approcci selezionati e la loro implementazione su Python. Infine, verranno mostrati i risultati ottenuti presentando un duplice confronto: uno che riguarda i vari metodi proposti, l'altro con l'obiettivo di rapportare le prestazioni raggiunte allo stato dell'arte presente in letteratura. È necessario sottolineare, però, che l'obiettivo del progetto è incentrato prevalentemente sull'analisi della prima comparazione, motivo

per il quale, anche per l'insufficiente potenza computazionale, alcune tecniche di miglioramento delle performance che esulano dallo specifico problema del learning to rank non sono state implementate. Questo dataset rappresenta un benchmark per il learning to rank e su di esso esistono tantissimi studi in letteratura. La possibilità di poter confrontare i risultati ottenuti con differenti strategie di implementazione è stata un fattore nella scelta del dataset da utilizzare.

3.2.1 Il dataset: MSLR-WEB10K

I dati presi in esame derivano dal dataset chiamato MSLR-WEB10K. Quest'ultimo è un insieme di dati fornito dalla Microsoft presente in due varianti, la 10K e la 30K, dove la differenza riguarda il numero di query considerate. Entrambi sono costruiti come riferimento per studi di machine learning relativi a simulazioni di motori di ricerca. Le informazioni, infatti, sono ottenute da Microsoft Bing e la stessa Microsoft fornisce una descrizione riguardo a cosa descrivono le diverse features considerate. La scelta di utilizzare il 10K anziché il 30K è dipesa da un mero limite computazionale, in quanto con i mezzi utilizzati è stato possibile trattare solamente le 10.000 query del 10K, non le oltre 30.000 del 30K.
I dati di MSLR-WEB10K sono forniti in formato tabellare, dove ogni riga rappresenta una coppia query-documento e ad ognuna delle 10.000 query sono associati, in media, 120 documenti. Ciascun elemento del dataset è rappresentato da un vettore di 138 componenti: la prima entrata è un intero che corrisponde al punteggio di rilevanza del documento; le altre colonne sono memorizzate nel formato *indice:valore*, di cui la prima caratterizza la query a cui il relativo documento corrisponde, mentre le restanti determinano le 136 caratteristiche attraverso le quali ogni documento è rappresentato. Per quanto riguarda l'etichetta, essa assume valori interi da 0 a 4, i quali indicano, rispettivamente, irrilevanza ed una perfetta corrispondenza tra documento e query. Ciò significa che il problema può essere ricondotto a un caso di classificazione multipla. Le features, invece, cercano di cogliere vari aspetti del documento ed infatti alcune dipendono solo dall'esempio stesso, altre dalla query a cui sono associate ed altre ancora dalla coppia documento-query. Nonostante la Microsoft dia una spiegazione riguardante ciascuna caratteristica, alcune informazioni legate alle visite dei siti web dipendono da dati privati della Microsoft e quindi assumono un significato poco chiaro. Dalla tabella presente nell'appendice A descrittiva tutte le features, è possibile notare che alcune di esse coincidono proprio con le tecniche non di machine learning descritte nel paragrafo 2.1, come la TF-IDF, BM25 e PageRank, giustificando così i ragionamenti precedentemente formulati.
Inoltre, il dataset si presenta già partizionato dai ricercatori della Microsoft. Difatti, esso è stato suddiviso in 5 parti contenenti 2000 query ciascuna, di cui tre costituiscono il training set, una il validation set e l'ultima il test set. In aggiunta a ciò, il dataset è composto da 5 cartelle in cui le 5 suddivisioni sono rimescolate tra i tre sottoinsiemi, così che ognuna venga utilizzata tre volte per addestrare i dati, una volta come validation per il settaggio dei parametri ed una quinta volta come dati di test. In questo modo, il dataset è già pronto per poter applicare la 5-fold cross validation descritta nel primo capitolo.
Dopo aver scaricato il dataset così costituito, grazie all'utilizzo della libreria *Pandas*, sono state effettuate operazioni di pre-processing, in modo da ottimizzare il formato con cui memorizzare e, successivamente, leggere i dati. Rispetto alla suddivisione proposta dalla Microsoft è stata utilizzata solamente la fold 1, senza effettuare, per questioni di tempo,

la cross-validation, in quanto l'attenzione si è rivolta allo sviluppo e la successiva comparazione dei metodi di learning to rank. Come prima cosa, in tutti gli insiemi sono state separate le etichette dalle features, caricandole in variabili distinte denominate, rispettivamente, X e Y. Per ognuna delle 10 variabili ottenute sono state poi effettuate le seguenti operazioni.
Innanzitutto sono stati raggruppati tra loro i documenti relativi alla stessa query sfruttando il corrispettivo valore della prima colonna. In questo modo si è potuto conteggiare il numero di documenti associati ad ogni query, quantità che sono state memorizzate in una nuova variabile separata. Inoltre, essendo a conoscenza di tale valore, è stato possibile conoscere a quale query ogni documento fosse associato semplicemente ordinando i dati in base all'*id* della query e conteggiando le righe visualizzate, un'operazione effettuata utilizzando *mergesort* come algoritmo di ordinamento. Questa scelta si spiega tenendo presente che, a differenza di altri, l'algoritmo di ordinamento selezionato è stabile, ossia ripetendo la procedura esso è in grado di mantenere lo stesso ordinamento anche tra oggetti caratterizzati dallo stesso valore. Un'evidenza importante perché, in caso contrario, effettuando le stesse operazioni anche sulle etichette, si sarebbe potuta determinare una incongruenza tra label e relativo documento. Effettuata tale procedura, si è reso possibile eliminare la colonna relativa all'*id* della query. Infine sono stati ripuliti i dati eliminando la componente *indice:* e trasformando il *valore* in un float. L'*indice*, infatti, è ora accessibile semplicemente dall'indice della colonna della feature, una trasformazione che consente di velocizzare, in seguito, la lettura dei dati e le successive operazioni effettuate dalla macchina.
A questo punto, i dati sono pronti per essere salvati ed utilizzati nelle varie procedure.

3.2.2 Pointwise + MSE

Sfruttando le variabili appena costruite, sono stati proposti quattro possibili metodi di learning to rank. Il primo, il più diretto da implementare, è l'approccio pointwise. In questo caso, infatti, non c'è bisogno di modificare il dataset, fatta eccezione per la standardizzazione rispetto al training set. Tale operazione è effettuata rispetto al training set in quanto questi sono i dati che realmente la macchina ha a disposizione. Ciò significa che gli elementi dell'intero dataset sono prima sommati alla media e, successivamente, divisi per la deviazione standard del training set, così da avere tutti i dati disposti secondo una distribuzione normale caratterizzata dalle stesse variabili. In questo modo la macchina è in grado di trovare più efficientemente schemi comuni tra i dati, garantendo così un miglioramento delle performance. A differenza delle altre strategie migliorative che esulano dal learning to rank, questa è stata attuata perché da un punto di vista teorico alcuni modelli, per essere utilizzati, richiedono che i dati assumano una distribuzione Gaussiana. In seguito all'attuazione di tale operazione è possibile progettare direttamente il modello, la cui procedura è riassunta nell'algoritmo 6.
Il primo passaggio consiste nell'addestrare una rete neurale sfruttando i dati presenti nel training set e nel validation set. Poiché in tutti i modelli che verranno presentati sarà utilizzata la medesima rete neurale, così che il confronto possa concentrarsi esclusivamente sulle differenze dovute all'utilizzo di un metodo piuttosto che un altro, senza che il modello incida, la struttura della rete adottata sarà spiegata nel dettaglio in seguito. A questo riguardo, le uniche differenze, da un metodo all'altro, risiedono nella scelta della funzione loss e nella selezione della funzione di attivazione e della dimensione dell'output

Algorithm 6 Algoritmo della procedura Pointwise+MSE

Require: dataset di partenza
Addestramento: addestra la rete neurale utilizzando come funzione loss la MSE sul training set e sul validation set
Predizioni: effettua le predizioni del test set con il modello addestrato precedentemente
Imposta le soglie come valori intermedi delle relevances
Converti le predizioni alle releveances intere a seconda dell'intervallo in cui esse si trovano
Valutazione: Calcola le metriche di valutazione
Output: NDCG@5, NDCG@10, NDCG@30 e Spearman correlation

finale.
In particolare, il caso pointwise può essere interpretato come un problema di classificazione multipla, in cui il modello è addestrato per predire con esattezza la classe di ogni documento sconosciuto. Per raggiungere questo scopo, il modello lavorerà in due passi: innanzitutto, la rete sarà addestrata a risolvere un problema di regressione, successivamente il risultato ottenuto, un numero reale, verrà arrotondato ad un intero tra 0 e 4 così che l'output finale corrisponda ad una classe specifica, risolvendo pertanto il problema di classificazione multipla. Adottando lo schema presentato nella sezione 2.4.1, la loss utilizzata dalla rete neurale è la *MSE*, la *Mean Squred Error*, la cui implementazione si trova all'interno di *Keras*, una libreria di alto livello creata su *TensorFlow*. Quindi, poiché l'output della rete deve essere un solo scalare reale compreso nell'intervallo delle relevances, la dimensione di output è fissata pari a 1 e la funzione di attivazione scelta è la *ReLU*. Così facendo, per le proprietà della funzione di attivazione scelta, il risultato, che già nei passaggi intermedi è forzato ad essere compreso tra 0 e 4 dai valori delle etichette, è ancora un valore che rientra in tale intervallo.
La rete addestrata, quindi, è utilizzata per predire i documenti del test set. Tali predizioni, però, non sono ancora valori interi corrispondenti alle etichette. Per questo motivo si fissano come soglie i punti intermedi tra i valori delle etichette, cioè 0.5, 1.5, 2.5 e 3.5. Attraverso tali soglie si potranno convertire i risultati delle predizioni in classi di appartenenza, a seconda dell'intervallo tra soglie entro cui cade la predizione. Mantenendo le soglie proposte, la conversione coincide con l'arrotondamento del valore all'intero più vicino. Tuttavia, la presenza delle soglie è utile poiché, se, ad esempio, si vuole ottenere un problema di classificazione binaria, si può fissare un'unica soglia che distingue l'etichetta 0 dalla 1, oppure, a causa di eventuali sbilanciamenti del dataset, si può decidere di allargare o ridurre l'intervallo corrispondente ad ogni label.
Infine, fornendo ora anche le etichette corrette del test set, è necessario confrontarle con le predizioni ottenute per valutare le prestazioni del modello. In particolare, verranno calcolate come metriche sia la NDCG per tre differenti range di posizioni, ossia 5, 10 e 30, sia il coefficiente di correlazione di Spearman. Anche in questo caso, dato che per tutti i modelli questa valutazione avverrà allo stesso modo sulle stesse metriche, l'implementazione del calcolo delle metriche sarà affrontato più dettagliatamente in seguito.

3.2.3 Ordinal + BCE

Il secondo metodo utilizzato consiste nella regressione ordinale descritta nel paragrafo 2.4.3. Infatti, dato che le etichette corrispondono ad un certo ordine di rilevanza, è possibile adottare questa strategia. Il flusso dell'algoritmo implementato è descritto nell'algoritmo 7.

Algorithm 7 Algoritmo della procedura Regressione Ordinale+BCE

Require: dataset ordinal ottenuto mappando ogni etichetta nella sua rappresentazione ordinale, cioè, con K classi, $k \mapsto [\underbrace{1,\dots,1}_{k},\underbrace{0,\dots,0}_{K-1-k}]$

Addestramento: addestra la rete neurale utilizzando come funzione loss la BCE sul training set e sul validation set

Predizioni: effettua le predizioni del test set con il modello addestrato precedentemente

Calcola l'aspettazione delle etichette ordinali predette

Converti le predizioni alle relevances intere arrotando all'intero più vicino

Valutazione: Calcola le metriche di valutazione

Output: NDCG@5, NDCG@10, NDCG@30 e Spearman correlation

Nonostante anche questo algoritmo si classifichi come pointwise, è necessario modificare inizialmente il dataset, in particolare le label. I vettori delle features, infatti, ad eccezione della medesima standardizzazione precedente, rimangono invariati. Per quel che riguarda le etichette, invece, esse vengono mappate come descritto nello pseudocodice. In particolare, realizzandole mediante l'utilizzo di un dizionario, le trasformazioni effettuate sono date da

$$0 \mapsto [0,0,0,0], \quad 1 \mapsto [1,0,0,0], \quad 2 \mapsto [1,1,0,0], \quad 3 \mapsto [1,1,1,0], \quad 4 \mapsto [1,1,1,1] \tag{3.4}$$

Realizzate queste trasformazioni, i dati sono pronti ed è possibile iniziare l'addestramento della rete neurale. Poiché l'output dovrà assumere lo stesso formato dell'input, il modello dovrà predire e restituire 4 valori per ogni label, motivo per il quale la dimensione dell'output in questo caso è fissata a 4. Inoltre, dato che le predizioni dovranno essere comprese tra 0 e 1, la funzione loss utilizzata è la *BCE*, cioè la *Binary Cross-Entropy*, che, come già spiegato, è tipica dei problemi binari. Anche in questo caso si è utilizzato Keras per implementare la funzione loss. Di conseguenza, per forzare il modello a restituire valori compresi in quell'intervallo, la funzione di attivazione finale è la *sigmoide*. In questo modo, dopo aver effettuato le predizioni sul test set, ogni cella dell'output rappresenta una probabilità.

Nello specifico, il risultato di ogni cella k indica la probabilità secondo cui la macchina attribuirebbe l'elemento considerato ad una classe rappresentata con un valore superiore a k. Quindi, effettuando la sottrazione tra una cella e la successiva, si ottengono le probabilità di appartenenza di un oggetto a ciascuna classe, cioè

$$\begin{aligned} P(k=0) &= P(k \geq 0) - P(k>0) = 1 - P[0], \\ P(k=1) &= P(k>0) - P(k>1) = P[0] - P[1], \\ P(k=2) &= P(k>1) - P(k>2) = P[1] - P[2], \\ P(k=3) &= P(k>2) - P(k>3) = P[2] - P[3], \\ P(k=4) &= P(k>3) - P(k>4) = P[3] - 0 = P[3] \end{aligned} \tag{3.5}$$

Allora, calcolate le singole probabilità, per risalire al formato originale dell'etichetta puntuale si calcola il valore atteso delle classi, in quanto esso formalizza l'idea di valore medio di una variabile aleatoria. Dato che le classi assumono valori discreti, l'etichetta predetta sarà data da

$$y_{pred} = \mathbb{E}[X] = \sum_{i=0}^{4} x_i P(k = i) \tag{3.6}$$

Successivamente, per ottenere le predizioni intere è sufficiente arrotondare il risultato del valore atteso all'intero più vicino. A questo punto, è possibile confrontare le predizioni con le etichette reali attraverso la medesima computazione delle metriche di valutazione utilizzata per il precedente algoritmo.

3.2.4 ListMLE

Per quanto riguarda l'approccio listwise, l'algoritmo scelto è ListMLE, in particolare la sua variante p-ListMLE. Questa scelta è dipesa sia dalle vantaggiose proprietà descritte nel paragrafo 2.6.4, sia dal fatto che in numerose ricerche esso è stato implementato garantendo ottime prestazioni. Così facendo, si dispone di adeguati risultati di riferimento in grado di confermare o meno la bontà delle conclusioni ottenute in questo progetto. Dopo aver applicato la solita standardizzazione rispetto al training set, nell'algoritmo 8 sono descritti i passaggi chiave di questo metodo.

Algorithm 8 Algoritmo della procedura Listwise mediante ListMLE

Require: dataset listwise tridimensionale dove, con operazioni di padding o di sampling, si uguaglia il numero di documenti per ogni query
Addestramento: addestra la rete neurale utilizzando come funzione loss la ListMLE sul training set e sul validation set
Predizioni: effettua le predizioni del test set con il modello addestrato precedentemente
Valutazione: Calcola le metriche di valutazione, adeguando le procedure alla diversa dimensione delle etichette
Output: NDCG@5, NDCG@10, NDCG@30 e Spearman correlation

Come spiegato in precedenza, l'istanza di un approccio listwise coincide con l'intera raccolta di documenti associati alla medesima interrogazione. Ad ogni query, però, non corrisponde sempre lo stesso numero di esempi. Ciò origina dei problemi nella creazione della rete in quanto i layer sono costruiti per ricevere un dato con una grandezza prefissata. Una possibile soluzione, quindi, è stata trovata creando un iperparametro, detto *slate length*, che stabilisca un numero prefissato di documenti che devono essere associati ad ogni query. Seguendo le indicazioni suggerite nel repository presente in [32], utilizzato come riferimento per la scelta degli iperparametri in virtù del fatto che propone un analogo lavoro sullo stesso dataset, lo slate length è stato inizialmente posto pari a 240. Tuttavia, per questioni di insufficiente potenza computazionale, si è utilizzato il massimo intero in grado di consentire l'addestramento della rete, ossia 200.
In base al numero di documenti originali, sono state impiegate due modalità per portare tale quantità al valore desiderato. In caso essi siano presenti in numero minore, l'operazione adottata è il padding, in particolare lo 0-padding. *Padding* letteralmente significa

"imbottitura": infatti ciò che viene eseguito è l'aggiunta alla query di esempi caratterizzati da features completamente nulle, la cui etichetta viene posta pari a 0, dato che essi sono ovviamente irrilevanti. In caso contrario, si selezionano casualmente, tra tutti quelli disponibili, tanti documenti, e relative etichette, quanto vale lo slate length. Questa operazione è chiamata *sampling*. Tale selezione è affiancata da un accorgimento che può permette di evitare di scegliere solamente documenti irrilevanti. Infatti, se si sono pescati esempi appartenenti solamente alla classe 0, si controllano gli altri: se nella query ne esiste esclusivamente uno diverso da 0 ed etichettato con 1, allora esso rimpiazza, casualmente, uno di quelli selezionati; se, invece, ne sono presenti molti di più, si ripete la procedura finché non viene scelto almeno un elemento non irrilevante. Evidentemente, se per caso ad una data query sono associati tanti documenti quanti quelli richiesti, non è necessario effettuare alcuna operazione. In seguito, ciascuna istanza, cioè ogni query, è inserita all'interno del dataset completo. In questo modo il dataset assume la forma tridimensionale $n_{query} \times slate\text{-}length \times n_{features}$ ed è esplicitata la suddivisione delle interrogazioni. La stessa costruzione è eseguita contemporaneamente sulle etichette, che quindi assumono grandezza pari a $n_{query} \times slate\text{-}length \times 1$. Il dataset ora è pronto per essere utilizzato dalla rete neurale.
Nonostante vengano effettuate query per query, anziché documento per documento, le predizioni assumono la stessa forma degli output della rete pointwise. Difatti, il risultato per cui la rete è addestrata rimane la singola etichetta di ogni documento, anziché direttamente la permutazione corretta o, comunque, un'etichetta globale dell'interrogazione. Per questo motivo, quindi, la procedura proposta non è propriamente un processo listwise, ma piuttosto un metodo pointwise adattato per poter essere addestrato attraverso una loss listwise. La dimensione e la funzione di attivazione finale di output della rete, quindi, coincidono con quelle del metodo pointwise + MSE. Anche la rete è costruita con la stessa struttura, ad eccezione della diversa dimensione di input, dato che ora i vettori sono tridimensionali, e di un layer aggiuntivo finale. Questo layer ha solamente il compito di "appiattire" gli output della rete: i risultati 3D vengono trasformati da questo layer in oggetti 2D di dimensione $n_{query} \times (slate\text{-}length \times n_{features})$ comprimendo le ultime due dimensioni in una unica. Questo è necessario a causa dell'implementazione della loss ListMLE utilizzata e presente in [31], che richiede che gli input siano bidimensionali.
In seguito all'addestramento si procede con il calcolo delle predizioni, che avviene nella maniera usuale.
Infine, si possono calcolare le metriche. Infatti, anche se le etichette predette non corrispondono ai valori interi rappresentanti delle classi, in questo caso è stata adottata la strategia listwise. Ciò che conta, quindi, è l'ordinamento, mentre l'esatta assegnazione delle classi è irrilevante. Le metriche di valutazione, pertanto, saranno calcolate come per gli algoritmi precedenti, confrontando i punteggi ottenuti direttamente con le etichette reali, senza trasformazioni intermedie, ed adottando opportuni accorgimenti dovuti alle dimensioni dei vettori, accorgimenti che saranno spiegati nel dettaglio nella sezione dedicata all'implementazione del calcolo delle metriche.

3.2.5 Pairwise

L'algoritmo pairwise che è stato costruito si basa sul modello RankNet spiegato nel paragrafo 2.5.2. L'unica differenza è che, riguardo la strategia di calcolo delle nuove predizioni, viene lasciata libertà alla rete neurale di operare come più ritiene opportuno a seconda

dei dati di training di cui dispone. Nell'algoritmo 9 viene presentato lo schema generale utilizzato, tralasciando nello pseudocodice gli accorgimenti dovuti alla gestione dell'aumento del costo computazionale che gli algoritmi pairwise comportano.

Algorithm 9 Algoritmo della procedura Pairwise+BCE

Require: dataset pairwise dove l'istanza a coppie è ottenuta concatenando a due a due tutti i documenti di ogni query
Addestramento: addestra la rete neurale utilizzando come funzione loss la BCE sul training set e sul validation set
Predizioni: effettua le predizioni del test set con il modello addestrato precedentemente
Arrotonda le probabilità predette al valore binario
Per ogni query crea una matrice $n_{doc-per-query} \times n_{doc-per-query}$ salvando nel triangolo superiore le preferenze binarie tra un documento e l'altro
Completa la matrice riempiendo il triangolo inferiore con la predizione contraria rispetto al valore contenuto nella posizione simmetrica
Calcola il punteggio di ranking del documento singolo facendo la media della relativa riga della matrice appena costruita
Valutazione: Calcola le metriche di valutazione
Output: NDCG@5, NDCG@10, NDCG@30 e Spearman correlation

La problematica maggiore legata a questo metodo è stata certamente l'aumento del costo computazionale, rispecchiando le criticità che già erano state sottolineate da un punto di vista teorico. Si pensi, infatti, che dal dataset puntuale di partenza, che richiede 982 MegaByte per essere salvato sulla macchina, la costruzione pairwise ha originato un dataset dal peso di 229 GigaByte. Ciò significa che l'aumento della complessità è stato dell'ordine di grandezza di 10^3, e questo ha reso impossibile trattare il problema alla stregua dei precedenti metodi. La soluzione, quindi, è stata l'utilizzo di *chunk*, ossia strutture che permettessero di leggere i file una frazione alla volta, senza richiedere alla macchina di memorizzare localmente l'intero dataset. Ciò ha reso trattabile il problema, almeno teoricamente. All'atto pratico è stato infatti necessario ciclare su tutti i chunk per poter analizzare l'intero dataset, causando tempi di addestramento spropositati.
Come prima cosa, si costruisce lo standardizzatore basato sul training set originale, così che la media e la deviazione standard non dipendano dalle coppie, madai documenti singoli. Tuttavia, si costruisce innanzitutto il dataset pairwise, così che esso sia composto dai dati originali, e solo in seguito viene applicata la standardizzazione.
Per creare il dataset si concatenano le coppie di documenti e a ciascuna nuova istanza viene assegnata la relevance 1 se il primo documento ha rilevanza maggiore o uguale del secondo, relevance 0 in caso contrario, il tutto lavorando per blocchi affinché la macchina non vada *Out of Memory*.
Per quanto riguarda l'addestramento, anche tale operazione deve essere eseguita a scaglioni. Prima di tutto, quindi, si costruisce la struttura della rete analogamente ai casi pointwise. La dimensione dell'input, infatti, è bidimensionale, l'output è dato da un unico valore, motivo per cui la dimensione finale vale 1, e, dovendo predire delle probabilità, la funzione di attivazione finale è la sigmoide, come nell'algoritmo di regressione ordinale. Successivamente, per ogni elemento dei chunk di training e di validation, che sono degli iteratori, ogni singolo documento di ciascuna coppia viene prima standardizzato e, in seguito, utilizzato per addestrare la rete. La dimensione di ogni chunk è un iperparametro,

che in questo caso è stata fissata ad essere un millesimo dell'insieme corrispondente. Ciò significa che per addestrare completamente la rete serviranno 1000 iterazioni.
Allo stesso modo si ragiona per il test set riguardo alle predizioni. Quindi, per ogni chunk del test, prima si standardizzano le features degli elementi e poi vengono effettuate le predizioni. Queste predizioni assumono valore reale, perciò, impostando una soglia pari a 0.5, gli output della rete vengono mappati in valori binari a seconda che essi siano maggiori o minori della soglia. Ottenute le preferenze di coppia, si devono ora costruire i punteggi di ranking individuali.
Considerando solo le query che contengono più di un documento, in quanto in caso contrario non possono esistere coppie e non vi sarebbe nulla da ordinare, per ognuna di esse si costruisce una matrice di dimensione $n_{doc-per-query} \times n_{doc-per-query}$ inizializzata con soli 0. In questa matrice verranno salvati i risultati delle preferenze, ponendo in ogni cella la preferenza dell'elemento della riga rispetto a quello indicato dalla colonna. La diagonale principale, quindi, rimarrà pari a 0, perché è inutile attribuire una relazione tra un documento e se stesso, e il triangolo superiore potrà essere riempito con le predizioni binarie appena calcolate. Il triangolo inferiore, invece, dovrà essere costruito in maniera contraria, perché se la preferenza dell'esempio i rispetto al j-esimo è 1, allora quando si considera j rispetto a i la preferenza dovrà essere 0 e viceversa. Per completare la costruzione, quindi, si seleziona il triangolo superiore già costruito e tali valori si sottraggono a uno, così da ottenere i risultati inversi. In seguito, la matrice triangolare appena ottenuta si traspone e si aggiunge a quella iniziale. Dopo aver completato la matrice, quindi, il punteggio di ogni documento della riga i sarà dato dalla media calcolata su tutti i documenti della riga considerata, ad eccezione di quello appartenente alla diagonale principale.
Dopo aver ottenuto i punteggi si procede con il calcolo delle metriche, calcolo che riflette esattamente lo stesso schema utilizzato negli algoritmi precedenti, dato che diventano disponibili le singole valutazioni.

3.2.6 La rete neurale

In questo paragrafo verrà spiegata dettagliatamente la struttura della rete neurale utilizzata come modello predittore per la risoluzione del problema. La rete assume la stessa struttura per ogni algoritmo, fatta eccezione per le necessarie modifiche già illustrate nel caso della loss listwise. Inoltre, per la scelta degli iperparametri si è preso spunto, come per lo slate length, dal lavoro realizzato in [32], così che le differenze tra i risultati ottenuti dipendessero da meno fattori possibili.
Dopo aver fissato il seed, il generatore di numeri casuali, in modo da rendere replicabili i risultati ottenuti e dopo aver inizializzato i pesi della rete, si è creata la struttura vera e propria utilizzando il modello *sequential* di Keras. La rete utilizzata è una rete fully connected costituita da 5 layer nascosti. Le dimensioni dei layer sono [256,512,1024,512,256]: esse prima aumentano e poi diminuiscono, così da lavorare su uno spazio di complessità maggiore e trovare più legami, i quali saranno poi ricondotti all'output desiderato. Tra un layer e l'altro, inoltre, è applicato sia un dropout con probabilità di silenziamento dei neuroni pari a 0.3 per prevenire l'overfitting, sia un'operazione di normalizzazione dei risultati. Come funzione di attivazione, per ogni livello intermedio è utilizzata la ReLU. Le strutture delle reti utilizzate sono rappresentate nella figura 3.1.
Per addestrare la rete si è utilizzato l'ottimizzatore Adam, che sfrutta tecniche di discesa del gradiente stocastico, con un learning rate fisso pari a 0.001. Stabilito un massimo

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	35072
batch_normalization (BatchNormalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131584
batch_normalization_1 (BatchNormalization)	(None, 512)	2048
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1024)	525312
batch_normalization_2 (BatchNormalization)	(None, 1024)	4096
dropout_2 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 512)	524800
batch_normalization_3 (BatchNormalization)	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
batch_normalization_4 (BatchNormalization)	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 1)	257

Total params: 1,358,593
Trainable params: 1,353,473
Non-trainable params: 5,120

(a) *struttura della rete bidimensionale*

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 200, 256)	35072
batch_normalization (BatchNormalization)	(None, 200, 256)	1024
dropout (Dropout)	(None, 200, 256)	0
dense_1 (Dense)	(None, 200, 512)	131584
batch_normalization_1 (BatchNormalization)	(None, 200, 512)	2048
dropout_1 (Dropout)	(None, 200, 512)	0
dense_2 (Dense)	(None, 200, 1024)	525312
batch_normalization_2 (BatchNormalization)	(None, 200, 1024)	4096
dropout_2 (Dropout)	(None, 200, 1024)	0
dense_3 (Dense)	(None, 200, 512)	524800
batch_normalization_3 (BatchNormalization)	(None, 200, 512)	2048
dropout_3 (Dropout)	(None, 200, 512)	0
dense_4 (Dense)	(None, 200, 256)	131328
batch_normalization_4 (BatchNormalization)	(None, 200, 256)	1024
dropout_4 (Dropout)	(None, 200, 256)	0
dense_5 (Dense)	(None, 200, 1)	257
flatten (Flatten)	(None, 200)	0

Total params: 1,358,593
Trainable params: 1,353,473
Non-trainable params: 5,120

(b) *struttura della rete tridimensionale*

Figure 3.1a e 3.1b : A sinistra la rappresentazione della rete bidimensionale, a destra la rete tridimensionale utilizzata per la ListMLE dove si può notare la presenza del layer di appiattimento finale

di 100 epoche, per prevenire l'overfitting si è adottata anche la tecnica dell'early stopping, settando un valore di patience di 5, con riferimento al valore della loss calcolata sul validation set. Il processo di addestramento, quindi, termina dopo aver visionato 100 volte l'intero dataset, oppure dopo che per 5 epoche consecutive il risultato misurato sul validation set non è migliorato. Infine, come batch size, cioè come numero di elementi visti contemporaneamente dalla macchina, è stato scelto 64, in modo tale che non fosse nè troppo piccolo, rallentando il processo di addestramento ed impedendo alla macchina di avere visioni d'insieme da cui imparare, nè troppo grande, per evitare di incorrere in problemi di memoria o di overfitting.

3.2.7 Il calcolo delle metriche di valutazione

L'operazione di calcolo delle metriche di valutazione è pressoché identica per tutti e quattro gli algoritmi, in quanto ogni predizione viene convertita in un punteggio di ranking singolo, dato che questo è il formato delle etichette reali. Solitamente, per verificare la bontà delle predizioni di un modello di machine learning si utilizza l'*accuracy*, ossia la precisione "puntuale" delle relevances. Per ottenerla, è sufficiente contare quante sono le predizioni corrette, cioè quelle che combaciano con le etichette reali, e rapportare tale valore al totale delle predizioni effettuate. Tuttavia, è già stato spiegato che, nel caso particolare del learning to rank, ciò che conta non è la correttezza della singola predizione, ma quella dell'ordinamento finale, tant'è che in alcuni casi i valori da comparare con le etichette reali non rappresentano label delle classi di rilevanza, ma solamente punteggi di

ranking. Pertanto, tale metrica non viene restituita dalla macchina e la concentrazione è posta esclusivamente sulla NDCG e sulla correlazione di Spearman, le metriche spiegate nella sezione 2.2.
Per computare tali valori si fa affidamento a *Scikit-learn*, una libreria Python di apprendimento automatico, per quanto riguarda l'implementazione della NDCG, e a *SciPy*, un'altra libreria Python contenente algoritmi e strumenti matematici, per il calcolo della Spearman correlation. Dopo aver forzato i vettori delle etichette ad essere vettori colonna, passaggio necessario per utilizzare le implementazioni delle librerie sopraelencate, si analizzano le predizioni una query per volta. È già stato evidenziato, infatti, che queste metriche agiscono su una singola query e per ottenere una valutazione globale del modello è quindi necessario calcolarne la media, motivo per il quale i singoli valori saranno salvati all'interno di liste inizialmente vuote. Un primo controllo riguarda il numero di documenti presenti in ogni query: se è presente un solo oggetto, la NDCG non è in grado di valutare la correttezza di un ordinamento che non esiste, per cui queste interrogazioni vengono scartate. Successivamente, grazie alle apposite funzioni delle precedenti librerie, si calcolano e memorizzano nelle liste i valori delle metriche. Il risultato finale, quindi, sarà ottenuto dalla media di tali valori. Tuttavia, per quanto riguarda la Spearman correlation è necessario effettuare un ulteriore controllo prima di poter calcolare la media dei singoli risultati. Difatti, situazioni in cui ad una query sono associati più documenti tutti con la stessa etichetta, comportano un valore di correlazione di Spearman pari a *Nan*, eventualità che, se si effettua la media, invalida i restanti valori. Di conseguenza, prima si cercano e si eliminano eventuali *Nan* all'interno della lista della correlazione e solo in seguito si calcola la media.
L'unica differenza che riguarda il calcolo delle metriche nel metodo listwise dipende dal fatto che, in questo caso, ogni riga delle predizioni corrisponde direttamente ad una intera query. Le predizioni, infatti, non avranno più dimensione $n_{doc-totali} \times 1$ ma $n_{query} \times slate\text{-}length$. Ciò significa che non sarà più necessario dover suddividere le etichette nelle rispettive query prima di calcolare le metriche: basterà semplicemente scorrere tutto il vettore delle predizioni ed implementare le metriche in modo che ad ogni iterazione esse abbiano in input la riga analizzata, la quale innanzitutto sarà trasformata in un vettore colonna. Successivamente, il calcolo delle medie per la valutazione globale e la ricerca dei valori *Nan* saranno identici.

3.2.8 Risultati

Verranno ora analizzate le performance raggiunte dai metodi Pointwise + MSE, ListMLE ed Ordinal + BCE. L'algoritmo pairwise non è stato testato perché, a causa dell'ingrandimento del dataset che esso comportava, con la macchina a disposizione il tempo di addestramento si è rivelato essere enorme. Dato che in letteratura i metodi pairwise sul dataset MSLR-WEB10K ottengono risultati simili all'implementazione pointwise, è subito possibile sottolineare una critica significativa. Infatti, il metodo pairwise comporta uno sforzo computazionale 1000 volte maggiore senza garantire importanti miglioramenti, motivo per il quale risulta più conveniente adottare uno qualsiasi degli altri metodi.
Oltre alla Spearman correlation, per effettuare un confronto con i risultati della letteratura presenti in [32], le metriche scelte per valutare i risultati ottenuti sono state NDCG@5, tipicamente utilizzata per i dataset MSLR-WEB, NDCG@10 e NDCG@30. Come prima analisi, il confronto si limita alla comparazione dei metodi implementati nel progetto, con

riferimento, quindi, a MSLR-WEB10K.

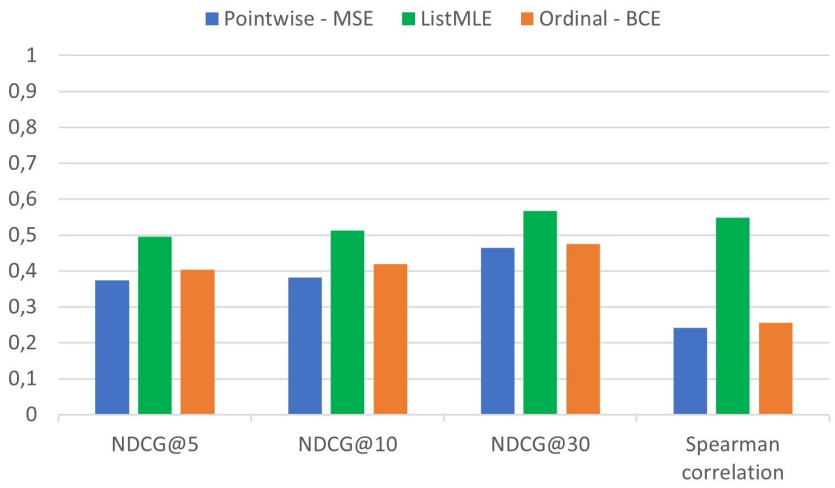


Figura 3.2: Accuratezza del ranking in termini della NDCG@n, n=5, 10, 30, e della correlazione di Spearman

Risulta evidente dal grafico riportato nella figura 3.2 che l'algoritmo migliore è quello che sfrutta la loss listwise ListMLE. Una conclusione positiva, in quanto conferma ciò che era stato studiato solo teoricamente nei capitoli precedenti, ossia che un approccio listwise è in grado di risolvere un problema di ranking nella maniera più appropriata. Ciò, infatti, è accertato anche dalla correlazione di Spearman, che misura la similarità tra tutto l'ordinamento predetto e quello reale. Inoltre, analizzando le performance degli altri due algoritmi, è evidente come l'algoritmo pointwise semplice ottenga le prestazioni peggiori, mentre l'accortezza di adattare il dataset ad un problema di regressione ordinale consente di migliorare i risultati pointwise, dato che il modello può attribuire probabilità di appartenenza alle classi in maniera indipendente tra loro. Tali differenze sono osservabili soprattutto quando si considera l'accuratezza delle primissime posizioni, confermando ancora una volta come la semplice visione pointwise, per quanto facile da implementare, non riesca a cogliere i principali aspetti che vanno tenuti in considerazione durante il ranking. Ovviamente, per diversi motivi precedentemente elencati, i risultati presentati non raggiungono lo stato dell'arte presente in letteratura, come emerge nella seguente tabella.

Loss	MSLR-WEB10K			MSLR-WEB30K - MLP			MSLR-WEB30K - Self-Attention		
	NDCG@5	NDCG@10	NDCG@30	NDCG@5	NDCG@10	NDCG@30	NDCG@5	NDCG@10	NDCG@30
Pointwise	0.3742	0.3825	0.4650	0.4824	0.5031	0.5629	0.5174	0.5340	0.5889
ListMLE	**0.4956**	**0.5123**	**0.5665**	0.4698	0.4914	0.5524	0.5020	0.5219	0.5794
Ordinal	0.4031	0.4198	0.4761	**0.4884**	**0.5102**	**0.5698**	**0.5300**	**0.5488**	**0.6019**

Tabella 3.1: Raccolta dei risultati del progetto e dell'articolo [32]

Qui sono rappresentati, rispettivamente, i risultati ottenuti nel progetto, cioè sul dataset MSLR-WEB10K, quelli ottenuti nell'esperimento spiegato in [32] mediante l'utilizzo di una rete neurale *Multi-layer perceptron*, cioè una rete analoga a quella descritta nel paragrafo 3.2.6, e quelli raggiunti dagli stessi autori sfruttando i nuovi studi sull'*attention* accennati nella sezione 2.6.5. Va sottolineato che ci sono differenze tra i due esperimenti,

in quanto in quello descritto nell'articolo il dataset è l'MSLR-WEB30K e soprattutto, alcune sottili differenze di implementazione sono certamente presenti, di cui la più vistosa risiede nel fatto che l'approccio pointwise studiato nel caso del 30K è costruito utilizzando come funzione loss la RMSE, anziché la MSE. Difatti, mentre i valori del metodo listwise sono sostanzialmente simili, i risultati raggiunti dagli altri due approcci sono migliori nel secondo esperimento. Questo può essere spiegato dal fatto che differenti implementazioni possono portare a diversi risultati, nonostante esse condividano le stesse fondamenta teoriche. Ciò dimostra che, con opportuni miglioramenti, ogni algoritmo può ottenere notevoli incrementi di performance. In particolare, utilizzando l'*attention* il modello ordinal migliora il suo rendimento significativamente, molto di più rispetto ai piccoli vantaggi che si ottengono nei due restanti casi.

3.3 Conclusioni

Lo studio che riguarda il learning to rank, pertanto, è lontano dall'essere completato, perché per ogni metodo si possono ancora scoprire nuovi accorgimenti specifici in grado di esaltarne le proprietà e ciò non può che garantire un miglioramento delle prestazioni, che certamente avverrà nel prossimo futuro. Peraltro, lo sviluppo degli studi di learning to rank non si limiterà esclusivamente al miglioramento degli algoritmi.
Un altro aspetto da tenere in considerazione che può comportare, allo stesso tempo, vantaggi computazionali ed incremento delle performance, è lo studio delle features. Infatti, ancora non si è riusciti a determinare quali siano le caratteristiche principali che, indipendentemente dal dataset utilizzato, garantiscano un migliore addestramento del modello. Spesso vengono considerate tutte le features raccolte nei precedenti studi e solo in seguito si verifica da quali di esse la macchina ha tratto più informazioni. La mancanza di features estremamente significative, però, può causare ridondanza e incapacità di generalizzazione nella macchina. Allo stesso modo, riempire i dataset di features richiede maggiori sforzi computazionali, sia per la loro memorizzarzione che per l'addestramento della rete. Un esempio in questo senso è dato proprio dai dataset MSLR-WEB, che con le loro 136 features costituiscono le raccolte con più caratteristiche tra tutti i dataset di riferimento del learning to rank. Ma, analogamente a ciò che succede negli approcci pairwise, se il problema aumenta di complessità vengono amplificate anche moltissime informazioni inutili. A tal proposito, infatti, sono già sorti studi, come il [20] del 2018, che hanno mostrato come sia possibile rimuovere almeno 40 features dal dataset MSLR-WEB senza perdere eccessivamente in termini di prestazione, quasi dimezzando così il costo computazionale. Questo accorgimento non è stato applicato nel progetto per poter effettuare un confronto più diretto con gli altri risultati della letteratura. Tuttavia, ciò è nuovamente una prova del fatto che, pur ottenendo già ottime performance, gli algoritmi di learning to rank e tutte le applicazioni che li sfruttano, non potranno che registrare nel tempo notevoli miglioramenti, forse ancora oggi impensabili, in grado di rivoluzionare la nostra vita.

Appendice A

Tabella descrittiva delle features del MSLR-WEB

Tabella A.1: Descrizione del significato delle features utilizzate nel dataset MSLR-WEB10K usato nel progetto.

Feature	No.	Description
covered query term number	1-5	How many terms in the user query are covered by the text. The text can be body, anchor, title, url [1] and whole document (for features 1 - 5 respectively, similarly below).
covered query term ratio	6-10	Covered query term number divided by the number of query terms.
stream length	11-15	Text length
IDF (inverse document frequency)	16-20	1 divided by the number of documents containing the query terms
sum of term frequency	21-25	Sum of counts of each query term in the document.
min of term frequency	26-30	Minimum of counts of each query term in the document.
max of term frequency	31-35	Maximum of counts of each query term in the document.
mean of term frequency	36-40	Average of counts of each query term in the document.
variance of term frequency	41-45	Variance of counts of each query term in the document.
normalized sum of stream length	46-50	Sum of term counts divided by text length.
normalized min of stream length	51-55	Minimum of term counts divided by text length.

[1] L'anchor text è il testo cliccabile di un link, l'URL di destinazione è l'indirizzo della pagina web a cui si verrà indirizzati quando si clicca sul collegamento.

Feature	No.	Description
normalized max of stream length	56-60	Maximum of term counts divided by text length.
normalized mean of stream length	61-65	Average of term counts divided by text length.
normalized variance of stream length	66-70	Variance of term counts divided by text length.
sum of tf*idf	71-75	Sum of the product between term count and IDF for each query term
min of tf*idf	76-80	Minimum of the product between term count and IDF for each query term
max of tf*idf	81-85	Maximum of the product between term count and IDF for each query term
mean of tf*idf	86-90	Average of the product between term count and IDF for each query term
variance of tf*idf	91-95	Variance of the product between term count and IDF for each query term
boolean model	96-100	**Unclear. Privately owned by Microsoft**.
vector space model	101-105	dot product between the vectors representing the query and the document. **The vectors are privately owned by Microsoft**.
BM25	106-110	Okapi BM25
LMIR.ABS	111-115	Language model approach for information retrieval (IR) with absolute discounting smoothing
LMIR.DIR	116-120	Language model approach for IR with Bayesian smoothing using Dirichlet priors
LMIR.JM	121-125	Language model approach for IR with JelinekMercer smoothing
number of slashes in URL	126	e.g., "ucsb.edu/pstate/people" has 2 slashes
length of url	127	The number of characters in the URL
Inlink number	128	The number of web pages that cite this web page
Outlink number	129	How many web pages this web cites.
PageRank	130	Evaluates the centrality of this web page based on web links over the Internet. This gives the success of Google.
SiteRank	131	Site level PageRank. E.g., "ucsb.edu/pstat" and "ucsb.edu/math" share the same SiteRank.

Feature	No.	Description
QualityScore	132	The quality score of a web page. The score is outputted by a web page quality classifier. **Privately owned by Microsoft**
QualityScore2	133	The quality score of a web page. The score is outputted by a web page quality classifier, which measures the badness of a web page. **Privately owned by Microsoft.**
Query-url click count	134	The click count of a query-url pair at a search engine in a period. **Collected and privately owned by Microsoft**.
url click count	135	The click count of a url aggregated from user browsing data in a period. **Collected and privately owned by Microsoft**.
url dwell time	136	The average dwell time of a url aggregated from user browsing data in a period. **Collected and privately owned by Microsoft.**

Bibliografia

[1] Martín Abadi. Tensorflow recommenders, 2015. Software available from tensorflow.org.

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. Learning a deep listwise context model for ranking refinement. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 135–144, 2018.

[3] Qingyao Ai, Xuanhui Wang, Nadav Golbandi, Mike Bendersky, and Marc Najork. Learning groupwise scoring functions using deep neural networks. 2019.

[4] Amir Atapour-Abarghouei, Stephen Bonner, and Andrew Stephen McGough. Rank over class: The untapped potential of ranking in natural language processing. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3950–3959. IEEE, 2021.

[5] Roger S Bivand, Edzer J Pebesma, Virgilio Gomez-Rubio, and Edzer Jan Pebesma. *Applied spatial data analysis with R*, volume 747248717. Springer, 2008.

[6] Christopher Burges, Robert Ragno, and Quoc Le. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, 19, 2006.

[7] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[8] Luis Campos. Learning to rank with tensorflow. *QuantDare*, 2019.

[9] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[10] Cristiano Casadei. Ia: Un po' di nozioni prima della pratica. *MaggioliDevelopers*, 2018.

[11] Cristiano Casadei. Le reti neurali ricorrenti. *MaggioliDevelopers*, 2020.

[12] Francesco Casalegno. Learning to rank a complete guide to ranking using machine learning. *Towards data science*, 2022.

[13] Kyle Chung. Introduction to learning to rank. 2019.

[14] David Cossock and Tong Zhang. Subset ranking using regression", booktitle="learning theory. pages 605–619, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[15] Na Dai, Milad Shokouhi, and Brian D Davison. Learning to rank for freshness and relevance. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 95–104, 2011.

[16] Maurice De Kunder. Geschatte grootte van het geïndexeerde world wide web. *Tilburg University*, 63, 2008.

[17] Ruoyao Ding. Literature survey for learning to rank. *Computer and Information Science Department University of Delaware*, page 6, 2009.

[18] Justin Evans. Learning to rank is good for your ml career - part 2: let's implement listnet! *Embracing the Random*, 2020.

[19] Daniel Godoy. Understanding binary cross-entropy/log loss: a visual explanation. *Towards data science*, 21, 2018.

[20] Xinzhi Han and Sen Lei. Feature selection and model comparison on microsoft learning-to-rank data sets. *arXiv preprint arXiv:1803.05127*, 2018.

[21] Leonid Kholkine, Thomas Servotte, Arie-Willem De Leeuw, Tom De Schepper, Peter Hellinckx, Tim Verdonck, and Steven Latré. A learn-to-rank approach for predicting road cycling race outcomes. *Frontiers in sports and active living*, 3:714107, 2021.

[22] Marius Köppel, Alexander Segner, Martin Wagener, Lukas Pensel, Andreas Karwath, and Stefan Kramer. Pairwise learning to rank by neural networks revisited: Reconstruction, theoretical analysis and practical performance. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*, pages 237–252. Springer, 2020.

[23] Bhuvana Kundumani. Simple neural network with bceloss for binary classification for a custom dataset. *Analytics Vidhya*, 2019.

[24] Yanyan Lan, Yadong Zhu, Jiafeng Guo, Shuzi Niu, and Xueqi Cheng. Position-aware listmle: A sequential learning process for ranking. In *UAI*, pages 449–458, 2014.

[25] Ping Li, Qiang Wu, and Christopher Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. *Advances in neural information processing systems*, 20, 2007.

[26] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[27] Andrea Minini. La differenza tra precisione (precision) e recall (richiamo) nel machine learning.

[28] William Stafford Noble. A quick guide to organizing computational biology projects. *PLoS computational biology*, 5(7):e1000424, 2009.

[29] Lawrence Page. Method for node ranking in a linked database. *USA Patent*, 6, 1997.

[30] Rama Kumar Pasumarthi, Sebastian Bruch, Michael Bendersky, and Xuanhui Wang. Neural learning to rank using tensorflow ranking: A hands-on tutorial. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 253–254, 2019.

[31] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2970–2978, 2019.

[32] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. Context-aware learning to rank with self-attention. *arXiv preprint arXiv:2005.10084*, 2020.

[33] Przemyslaw Pobrotyn, Tomasz Bartczak, Mikolaj Synowiec, Radoslaw Bialobrzeski, and Jaroslaw Bojar. Context-aware learning to rank with self-attention. *ArXiv*, abs/2005.10084, 2020.

[34] Przemyslaw Pobrotyn and Radoslaw Bialobrzeski. Neuralndcg: Direct optimisation of a ranking metric via differentiable relaxation of sorting. *ArXiv*, abs/2102.07831, 2021.

[35] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374, 2010.

[36] Tao Qin, Xu-Dong Zhang, De-Sheng Wang, Tie-Yan Liu, Wei Lai, and Hang Li. Ranking with multiple hyperplanes. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 279–286, 2007.

[37] Sebastian Raschka. *Machine Learning Q and AI.*

[38] Nick Sayer. Google code archive-long-term storage for google code project hosting. *XP055260798, Retrieved from the Internet [retrieved on 20160323]*, 2014.

[39] Jian Shi and Xin-Yu Tian. Learning to rank sports teams on a graph. *Applied Sciences*, 10(17), 2020.

[40] Jian Shi and Xin-Yu Tian. Learning to rank sports teams on a graph. *Applied Sciences*, 10(17):5833, 2020.

[41] Yoshihiko Suhara, Jun Suzuki, and Ryoji Kataoka. Robust online learning to rank via selective pairwise approach based on evaluation measures. *Information and Media Technologies*, 8(1):118–129, 2013.

[42] Yi Tay, Minh C Phan, Luu Anh Tuan, and Siu Cheung Hui. Learning to rank question answer pairs with holographic dual lstm architecture. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 695–704, 2017.

[43] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 77–86, 2008.

[44] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, 2007.

[45] Heather Turner, Jacob van Etten, David Firth, and Ioannis Kosmidis. Introduction to plackettluce, 2018.

[46] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 1313–1322, 2018.

[47] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, 2008.

[48] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, 2007.

[49] Xin Zhang, Lan Wu, and Zhixue Chen. Constructing long-short stock portfolio with a new listwise learn-to-rank algorithm. *Quantitative Finance*, 22(2):321–331, 2022.

[50] Xiaofeng Zhu and Diego Klabjan. Listwise learning to rank by exploring unique ratings. In *Proceedings of the 13th international conference on web search and data mining*, pages 798–806, 2020.

Ringraziamenti

Prima di tutto vorrei ringraziare la professoressa Silvia Bonettini, mia relatrice, la quale mi ha aiutato e supportato nella stesura di questa tesi, così come durante l'intero percorso magistrale. Allo stesso modo vorrei dire grazie a tutti i colleghi di Axyon, che mi hanno accolto facendomi sentire uno di loro. In particolare, un grazie sentito va certamente a Giovanni Davoli, che mi ha sempre aiutato rispondendo a tutte le mie domande supportandomi e, soprattutto, sopportandomi.

Vorrei inoltre ringraziare la mia intera famiglia, che mi è sempre stata accanto: nonni, zii e cugini, grazie per il sostegno, davvero. Un ringraziamento a parte lo meritano i miei genitori, Ermes e Stefania (l'ordine è puramente alfabetico, non sia mai...). So talvolta di essere una borsa che non vi dice mai grazie abbastanza volte, quindi questa è l'occasione adatta per rimediare.

La mia famiglia però non si limita a queste persone. Infatti vorrei dire grazie a tutti i miei amici e alle persone che mi vogliono bene, perché questo percorso parte da molto lontano e non sarebbe stato certamente lo stesso senza di voi.

Infine, ma non certamente per importanza, un enorme ringraziamento va a colei che mi affianca da due anni e mezzo a questa parte, sperando vivamente che questo possa essere solo l'inizio. Grazie Martina, per un mare (fuori) di motivi: dal tuo aiuto per gli esami e per questa tesi ma, soprattutto, per apprezzarmi così come sono e rendermi ogni giorno felice ed una persona migliore.