

MACHING LEARNING FOR IR

Il machine learning viene usato nei sistemi IR per fare un rank dei documenti ritrovati come risultato.

Modelli generativi

Assumono che i documenti sono generati da qualche modello

Modelli discriminativi

Stimano la probabilità che un documento appartenga a una certa classe (Rilevante o Non) solo guardando le feature del documento. Serve però un set di dati di training.

Il learning to rank non è proprio descrivibile come un problema di classificazione o regressione.

Qui bisogna mappare i documenti con un set ordinato di classi (Nei classificatori il set non è ordinato).

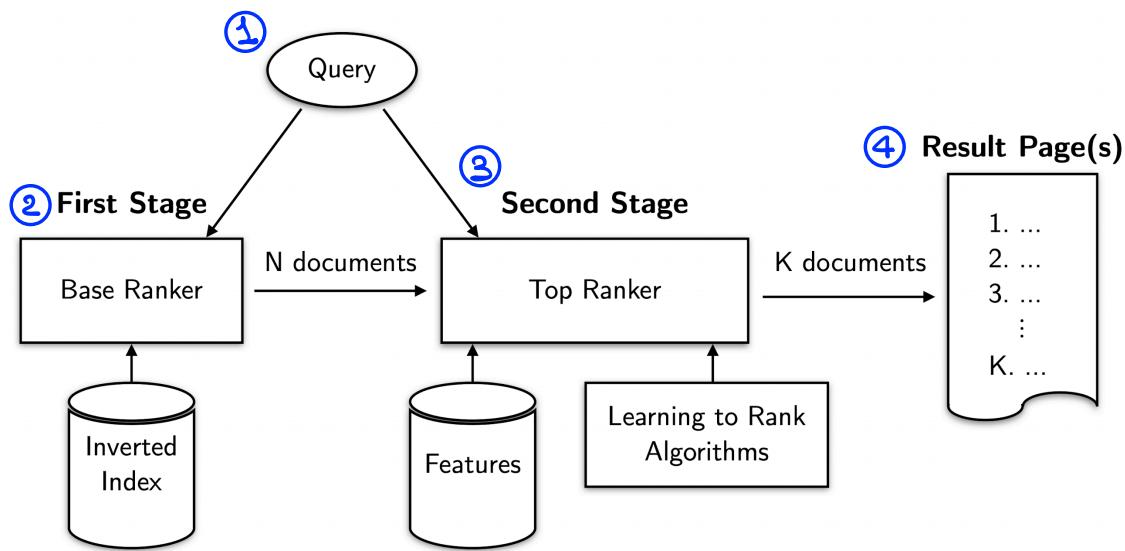
Questi modelli vengono allenati su dataset aventi un relevance assessment.

Questo approccio rende il sistema più efficace ma di contro colta l'efficienza e quindi le prestazioni in termini di velocità.

RANKING CASCADAS

Tipicamente, su grande scala (Web), il processo di ranking può essere visto come una serie di cascade (cascades):

- 1) Fare il rank di qualche documento
- 2) Passare i top-K al cascade successivo per perfezionare il rank



1) Ricezione della query

2) First Stage: si fa il rank con BM25 e l'inverted index, ricevendo così N documenti.

Note: In questa fase è importante massimizzare la **RECALL** degli N documenti, questo perché siamo interessati, in questa fase, a prendere il massimo numero di documenti rilevanti da passare al secondo stage.

3) Second Stage: Si usa un modello di machine learning per ricevere i top-k.

NOTA: In questa fase dobbiamo massimizzare la precision ($P@K$).

4) Ripetiamo i top-k documenti trovati

LEARNING TO RANK

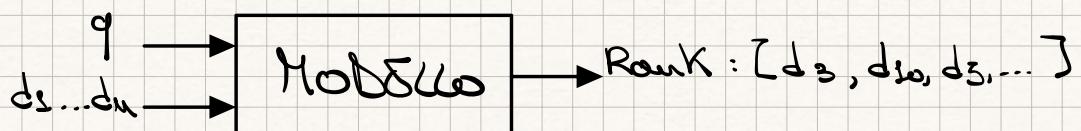
Per allenare un modello serve un dataset di training. A ogni query viene associata una lista di documenti:

- 1) I documenti sono descritti da feature reali
 $[f_{x_0}, f_{x_1}, f_{x_2}, f_{x_3}, \dots]$
- 2) Ogni documento ha una label y_{x_i} a indicare la rilevanza del documento per quella query:
 $y_{x_i} \in [0, 4]$

Abbiamo quindi per ogni query:

$$q_a \rightarrow [(d_4, y_{a4}), (d_2, y_{a2}), (d_3, y_{a3}), (d_8, y_{a8}), (d_{41}, y_{a41}), \dots]$$

$$q_b \rightarrow [(d_{99}, y_{b99}), (d_4, y_{b4}), (d_7, y_{b7}), (d_2, y_{b2}), (d_{11}, y_{b11}), \dots]$$



Tipi di feature

Soltanente si usano tre tipi di feature:

1) Query-document feature

Ovvio le metriche come BM25

2) Query-independent feature

Numero di out-link alle pagine web

Numero di in-link

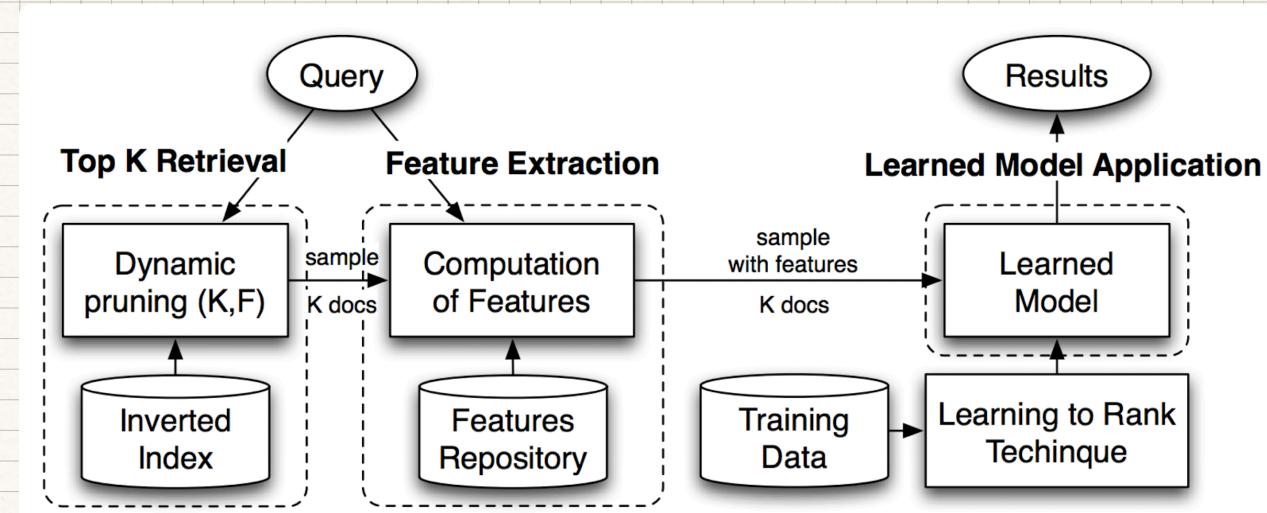
Lunghezza URL, titolo, score spam

3) Query-only

Numero di termini della query

Numero di stopword

Numero di entità, o termini che possono riferirsi a una pagina Wikipedia



Ovviamente, come tutti gli approcci di ML abbiamo bisogno di un **training set**, **validation set** e **test set**.

Per stimare la bontà di un modello, si usa una **loss function** da minimizzare sui dati di training. Esistono tre famiglie di **Learning Procedure** con le relative loss function associate:

1) POINT-WISE

Similmente ai modelli di classificazione o regressione cercano di rispondere a:

- Quanti documenti il modello ha correttamente valutato come rilevanti o non? (Classificazione)
- Quanto si distacca lo score previsto rispetto a quello della Label? (Regressione)

Le **limitazioni** sono che

- Considera ogni documento indipendentemente, invece di mettere i documenti rilevanti sopra quelli non rilevanti e quindi dipendendo da essi.
- Ignora il fatto che alcuni documenti sono associati alla stessa query e altri no
- Se il numero di docs ritornati varia molto da query a query; le query con il maggior numero di risultati sono agevolate dalla loss.
- La posizione in classifica dei documenti non è messa in conto nella loss function(s)

2) PAIR-WISE

In questo caso abbiamo un input $\langle q, (d_1, d_2) \rangle$, dove i due documenti hanno qualche preferenza fra loro (uno rilevante l'altro no, ma con più clicks dell'altro). L'idea è quella di minimizzare la loss basandosi sull'ordine nel classificare questi due documenti.

Problemi:

- Non sempre approssima la misura della loss a un numero reale.
- Il numero di copie di documenti da analizzare è dell'ordine quadratico rispetto al numero di risultati.

3) LIST-WISE

Considera l'intera classifica dei documenti, e incapsula una funzione di rank.

Lento ma efficace

LINEAR MODELS

Alcune tecniche di learning-to-rank usano la combinazione lineare delle feature

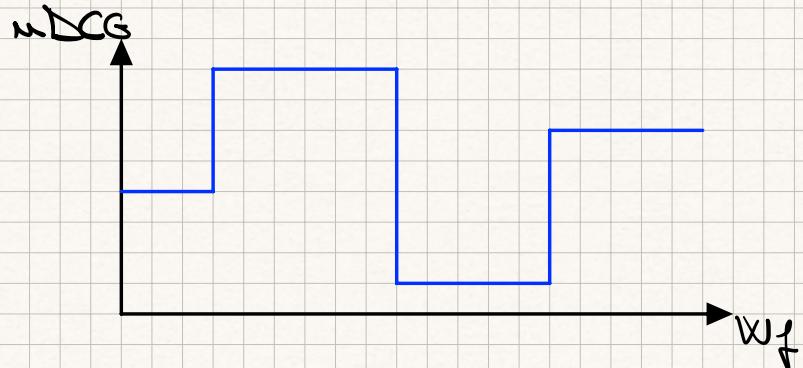
$$S(q, d) = \sum_{f \in \mathcal{F}} w_f f(q, d)$$

I modelli lineari fanno delle assunzioni

- 1) Le stesse feature sono necessarie a tutte le query.
- 2) Il modello deve essere una combinazione lineare delle feature
- 3) Se una feature ha grosso peso, allora è lo stesso per tutte le query

Come calcoliamo w_f per maximizzare la performance del sistema IR? Gradient descent.

Ma le metriche di valutazione di un sistema IR non sono continue rispetto ai pesi w_f , e non sono sempre differenziabili

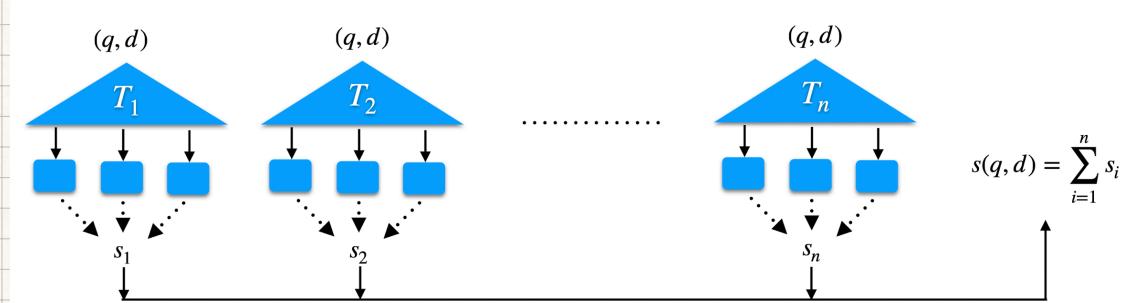


Le metriche (nDCG) non cambiano al cambiare delle scese di un documento, ma solo quando esso cambia posizione. C'è uno "swap" fra due documenti. La derivata del grafico è zero ovunque tranne nei punti di swap, dove è infinita. MAU! Saremo tecniche matematiche più avanzate della semplice oscura del gradiente.

TR35 Model

Abbiamo un albero regressivo che data la coppia q, d approssima la scorsa.

Creiamo una foresta di alberi atti a calcolare la scorsa finale.



Il fatto che negli alberi si possono personalizzare i "branch" rende il modello ad alberi più efficace di quello lineare.

Questi modelli sono **Point-wise** per natura, ma adattabili a **List-wise**