QuProBERT - BERT per la classificazione Query-Prodotto

Carmine Iannotti
Dipartimento di Informatica
Università degli Studi di Salerno

Mattia Limone
Dipartimento di Informatica
Università degli Studi di Salerno

***Abstract*—Garantire risultati rilevanti nella ricerca dei prodotti è un compito critico, in quanto influisce sulla capacità del cliente di trovare i prodotti desiderati nel breve termine, sulla percezione complessiva e la fiducia nel sistema di e-commerce nel lungo termine. Migliorare la qualità dei risultati tende a incrementare il coinvolgimento degli utenti con i motori di ricerca.**

Nonostante i recenti progressi nel campo dell'apprendimento automatico e del data mining, classificare correttamente gli articoli per una particolare query di ricerca dell'utente è una sfida di lunga data, che ha ancora un ampio margine di miglioramento.

In questo lavoro è stato utilizzato lo *Shopping Queries Dataset*, un grande set di dati composto da query e risultati di ricerca difficili avvenute su Amazon, rilasciato pubblicamente con l'obiettivo di promuovere la ricerca sul miglioramento della qualità dei risultati di ricerca.

Andremo a descrivere il dataset, il processo di pulizia dei dati ed a presentare una nuova rete neurale sequenziale come soluzione al problema della classificazione multiclasse dei risultati di una ricerca di prodotti in categorie di rilevanza.

***Index Terms*—NLP, KDD'22, BERT, Amazon**

I. Introduzione

La ricerca di un prodotto in un catalogo, o più in generale il task di restituire informazioni pertinenti la richiesta effettuata, è spesso inficiata dalla presenza di informazioni rumorose nei risultati, dalla difficoltà di comprendere l'intento della query e dalla eterogeneità dei possibili risultati da restituire. Tutto ciò rende il task di restituire prodotti pertinenti alla richiesta un task molto difficile. Quando si sviluppano modelli di apprendimento automatico per applicazioni di shopping online, è necessaria un'accuratezza estremamente elevata. A ciò si aggiunge che le scarse conoscenze del funzionamento del sistema da parte dell'utente limitano la sua esperienza, in particolare, la classificazione di ogni prodotto mostrato in risposta a una query dell'utente come pertinente o meno produce in genere risultati che hanno un effetto negativo. Ad esempio, per la query "Samsung S21", un caricabatterie per Samsung S21 è rilevante, irrilevante o una via di mezzo? In realtà, molti utenti digitano la query "Samsung S21" per trovare e acquistare un caricabatterie per lo stesso. Si aspettano semplicemente che il motore di ricerca comprenda le loro esigenze a partire da richieste molto generiche.

Per questo motivo, la rilevanza di un articolo viene suddivisa nelle seguenti quattro classi, utilizzate per misurare la rilevanza degli elementi nei risultati di ricerca:

- **Exact (E)**: l'articolo è rilevante per la query e soddisfa tutte le specifiche della query (ad esempio, "zaino in pelle uomo" che corrisponde a tutti gli attributi della query "Zaino Uomo Porta PC Pelle XL", come il materiale e le dimensioni).
- **Substitute (S)**: l'articolo è poco rilevante, cioè non soddisfa alcuni aspetti della query, ma può essere usato come sostituto funzionale (ad esempio, una powerbank per la query "caricabatterie").
- **Complement (C)**: l'articolo non soddisfa la query, ma potrebbe essere usato in combinazione con un articolo esatto (ad esempio, "Penna Sfera Colorate" per la query "astuccio").
- **Irrilevant (I)**: l'elemento è irrilevante o non soddisfa un aspetto centrale dell'interrogazione (ad esempio, "Scarpe da ginnastica" per una query "vestito elegante")

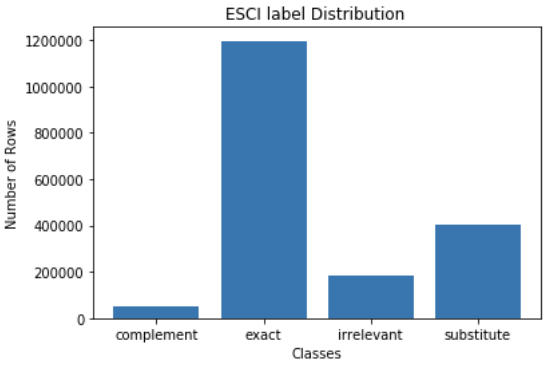


Fig. 1: Distribuzione delle Label

II. Lavori correlati

La generazione di rappresentazioni testuali efficaci per i tasks di classificazione è un'area di ricerca attiva che ha visto notevoli miglioramenti negli ultimi anni [1–3]. Quasi la totalità delle proposte presenti in letteratura è basata su BERT [3], un modello multi-strato basato sulla rappresentazione fornita da un encoder bidirezionale attraverso l'uso dei Transformer [4].

Garakani et al. [5] hanno sviluppato un modello capace di definire la correlazione fra due frasi utilizzando l'output cross-encoder di BERT come sequenza di input, sequenza che è stata poi segmentata in sotto-parole usando l'algoritmo WordPiece [6] per poi usare il modello BERT-base come classificatore. Questo può essere considerato un approccio valido nei casi

Simbolo	Descrizione
NLP	Natural Language Processing
E	Exact
S	Substitute
C	Complement
I	Irrilevant
W	Weights
Q	Query
K	Key
V	Value
IR	Information Retrieval
MAP	Mean Average Precision
MLP	Multi Layer Perceptron
LSTM	Long short-term memory
ReLU	Rectified Linear Unit
LR	Learning Rate

Tabella I: Simboli utilizzati nel documento.

in cui si ha necessità di sviluppare modelli a basso costo di sviluppo e con scarse risorse di calcolo.

Dahlmann e Lancewicki [7] sono andati a valutare le capacità di predizione delle più popolari versioni specializzate di BERT sul task di classificazione binaria dei prodotti utilizzando il dataset QTR (query, titolo del prodotto e grado di rilevanza) e in secondo luogo hanno effettuato il fine-tuning degli stessi e rivalutato le capacità predittive, così facendo hanno dimostrato che un modello di BERT ottimizzato per il compito di classificazione sul dataset QTR fornisce prestazioni nettamente superiori rispetto alle altre soluzioni presenti nei maggiori siti di e-commerce. Così facendo hanno dimostrato che non è necessario utilizzare versioni di BERT eccessivamente complesse e che versioni molto più leggere e pronte alla lancio come BERT-student e BertBiLSTM ottengono risultati simili alle altre versioni più complesse.

Kocián et al. [8] hanno sviluppato un modello siamese costituito da due branch di ELECTRA [9] che prendono in input rispettivamente una query ed una descrizione di un prodotto con cui effettuare il matching generando due rappresentazioni vettoriali dell'input tra le quali sono state calcolate la similarità del coseno e la distanza euclidea, il risultato è infine affidato ad una rete feed-forward che svolge il compito di classificatore. Il modello sebbene molto più complesso rispetto agli altri precedentemente citati rappresenta lo stato dell'arte per la classificazione dei prodotti in lingua ceca ottenendo una precisione migliore del 3.8% rispetto alle altre soluzioni prsenti in letteratura.

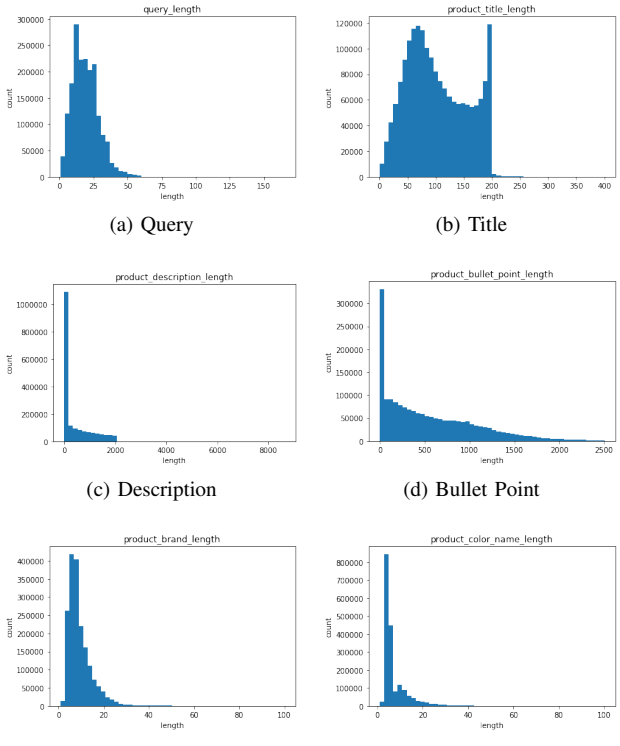
III. Dataset

In questo lavoro è stato utilizzato lo *Shopping Queries Dataset* [10]. Un set di query di ricerca pubblicato con l'obiettivo di sviluppare nuovi modelli di NLP per la comprensione della relazione semantica tra query e prodotti.

Per ogni query, il dataset fornisce un elenco di fino a 40 risultati, insieme ai loro giudizi di rilevanza ESCI (Exact, Substitute, Complement, o Irrelevant) che indicano la rilevanza del prodotto rispetto alla query.

Il dataset fornito è multilingue, in quanto contiene query in inglese, giapponese e spagnolo. Con questi dati proponiamo di classificare le coppie query/prodotto nelle categorie E,S,C, o I. Questa raccolta ha alcune caratteristiche che la rendono particolarmente interessante per la ricerca nel campo dell'information retrieval e della classificazione.

In primo luogo, deriva da clienti reali che cercano prodotti reali online, inoltre fornisce sia ampiezza (un gran numero di interrogazioni, in tre lingue) sia profondità (20 risultati per interrogazione) a differenza di altre raccolte esistenti che tendono a fornire o l'ampiezza o la profondità, ma non entrambe. Infine, tutti i risultati sono stati etichettati manualmente con etichette, che descrivono lo stato di rilevanza data la query di ricerca, infine le query non sono state campionate in modo casuale, ma piuttosto sottoinsiemi di query sono stati selezionati specificamente per garantire l'eterogeneità.



(a) Query (b) Title

(c) Description (d) Bullet Point

(e) Brand (f) Color Name

Fig. 2: Istogrammi per attributo

Ogni istanza presente nel dataset fornito è composta da una coppia query-risultato alla quale è stato associato un ESCI label, ed altri attributi riguardanti il prodotto, nel seguente ordine:

- **example_id**: identificativo univoco dell'istanza
- **query**: stringa rappresentante la query del cliente
- **query_id**: identifactivo univoco della query
- **product_id**: identificativo univoco del prodotto corrispondente ad un prodotto realmente presente nel catalago Amazon
- **product_title**: titolo del prodotto esattamente come nel catalago Amazon
- **product_description**: descrizione del prodotto esattamente come nel catalago Amazon

Attributo	query	product_id	title	description	bullet_point	brand	color_name
MEDIA	19.52	10.00	103.33	396.06	560.15	9.03	6.82
STD	10.05	0.0	53.79	584.46	517.23	5.61	6.49
MIN	1.00	10.00	1.00	1.00	1.00	1.00	1.00
25%	12.00	10.00	60.00	3.00	121.00	6.00	3.00
50%	18.00	10.00	94.00	3.00	429.00	7.00	5.00
75%	25.00	10.00	148.00	667.00	888.00	11.00	8.00
MAX	165.00	10.00	400.00	8640.00	2503.00	100.00	100.00

Tabella II: Statistiche del Dataset

- **product_bullet_point**: scheda tecnica del prodotto esattamente come nel catalago Amazon
- **product_brand**: stringa rappresentante il brand del prodotto
- **product_color**: stringa rappresentante il colore del prodotto, solo se è applicabile
- **product_locale**: stringa rappresentante il catalogo di provenienza (US per il catalogo inglese, JP per il catalogo giapponese, ES per il catalogo spagnolo)
- **esci_label**: la label di output da predire

IV. Preprocessing

Nei task di NLP, così come nella totalità dei task di Machine Learning, la fase di preprocessing è cruciale affinchè il modello di predizione finale possa correttamente generalizzare.

La pulizia dei dati applicata al dataset fornito è stata così svolta:

- Rimozione del rumore
  - Rimozione dei tags HTML
  - Rimozione della punteggiatura ininfluente
  - Rimozione degli URLs
  - Rimozione delle stopwords
  - Lower casing
  - Lemmatization
- Data Imputation
- Tokenization

A. Rimozione del rumore

Al fine di rimuovere tutte quelle informazioni che potessero generare eccessivo rumore si è proceduto con l'eliminare tutto ciò che a nostro avviso non risultava rilevante per comprendere la semantica di un testo.

1) Rimozione URLs e tags HTML: I dati relativi al prodotto restituito da una query è stato prelevato tramite un Web Scraper. Di conseguenza gli attributi product_description e product_bullet_point risultavano completi dei relativi tag HTML di impaginazione. Sfruttando la libreria $BeautifulSoup$ [11] che fornisce un HTML Parser completo ed ottimizzato è stato possibile rimuovere tutti i tags e gli URLs.

2) Rimozione punteggiatura e stopwords: Quando si ha a che fare con testi di grandi dimensioni e si vogliono eseguire compiti specifici come il calcolo della similarità tra documenti, la classificazione o la clusterizzazione si preferisce rimuovere dal testo la punteggiatura ininfluente e le stopwords.

Si sente spesso parlare di "stop words", "stop word list" o anche "stop list". Le stop words sono fondamentalmente un insieme di parole comunemente usate in qualsiasi lingua, non solo in inglese.

Il motivo per cui la rimzione delle stop words è fondamentale per molte applicazioni è che, se eliminiamo le parole di uso comune in una determinata lingua, possiamo concentrarci sulle parole importanti.

3) Lower Casing: In questa fase si è semplicemente provveduto a trasformare l'intero testo in minuscolo. Ciò è stato effettuato affinchè a parole come $Smartphone$ e $smartphone$ non venisse attribuito un token differente andando così a semplificare i dati su cui il modello sarà allenato.

4) Lemmatization: Per motivi grammaticali, i documenti utilizzano forme diverse di una parola, come $organizziamo$, $organizzo$ e $organizzai$. Inoltre, esistono famiglie di parole derivate con significati simili, come $democrazia$, $democratico$ e $democratizzazione$. In molte situazioni è utile che la ricerca di una di queste parole restituisca documenti che contengono un'altra parola dell'insieme.

L'obiettivo della lemmatization, dunque, è quello di ridurre le forme flessionali e talvolta derivate di una parola riconducendole ad una forma base comune. Ad esempio:

$$am, are, is \rightarrow be$$

$$gattino, micio, gatta \rightarrow gatto$$

$$pequeño, niño, niña \rightarrow niño$$

La lemmatization è basata sull'uso di un vocabolario linguistico e di un'analisi morfologica delle parole, con l'obiettivo di rimuovere solo le terminazioni flessionali e di restituire la forma base o dizionario di una parola, che è nota come lemma. La lemmatization, a differenza dell'approccio stemming, modifica solo le diverse forme flessionali di un lemma.

Per effettuare queste operazioni ci siamo avvalsi della libreria $nltk$ che già supporta sia l'inglese che lo spagnolo che il giapponese.

B. Data Imputation

Le operazioni fin qui effettuate hanno permesso di ridurre sensibilmente la lunghezza media dei dati riguardanti un generico prodotto.

Nonostante ciò alcuni attributi risultavano ugualmente di dimensioni proibitive. Di conseguenza sono stati scelti i soli attributi che maggiormente permettono di definire un prodotto all'interno di un catalogo che sono titolo, colore, taglia e brand.

Modello	M1	M2	M3	M4	M5	M6	M7	M8
Learning Rate	5e-5	5e-5	4e-5	4e-5	3e-5	3e-5	1e-5	1e-5
Learning Rate Warmup	10000 steps	10000 steps	10000 steps	10000 steps	10000 steps	10000 steps	10000 steps	10000 steps
L2 Weight Decay	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
β**1**	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
β**2**	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Dropout on Dense Layer	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Batch size	16	32	16	32	16	32	16	32
Epochs	4	4	4	4	4	4	4	4

Tabella III: Parametri di Fine-Tuning [3]

Inoltre in circa 6000 istanze fornite alcuni di questi campi risultavano nulli. Si è proceduto dunque ad attuare una fase di data imputation al fine di avere un dataset denso.

Abbiamo quindi effettuato il fine-tuning del modello T5 [12] con le istanze complete sfruttando il campo descrizione del prodotto come testo dal quale estrarre i dati mancanti.

Così facendo abbiamo potuto sfruttare le capacità di predizione del modello riallenato per riempire i valori nulli.

C. Tokenization

Dato che il modello proposto è un versione modificata di BERT, si è utilizzato il Tokenizer nativo per la generazione dei dati di input. Il Tokenizer è basato sulla semplice scomposizione delle frasi in sottoparole mantenendo informazioni del contesto.

Prendiamo ad esempio il seguente input:

- Query: Iphone 13
- Prodotto: Iphone 13 Pro Max, Space Gray

In prima fase il Tokenizer provvederà ad inserire alcuni token speciali:

- `[CLS]`: sta per Classification e serve a rappresentare la classificazione a livello di frase.
- `[SEP]`: sta per Separator e serve per indicare la separazione tra due frasi o il termine della frase

L'utilizzo di questi token è necessaria in quanto BERT per il task di NSP, ovvero per la comprensione del legame fra una frase A ed una frase B, è stato allenato e dunque prevede come input per il suo utilizzo, un istanza così formata:

`[CLS]` FRASE A `[SEP]` FRASE B `[SEP]`

Di conseguenza l'istanza in esempio diventerà:

`[CLS]` Iphone 13 `[SEP]` Iphone 13 Pro Max, Space Gray `[SEP]`

A partire da questo verranno generati i seguenti array:

- Token Embeddings: ids di input per l'analisi semantica
- Segment Embeddings: array che rappresenta la separazione delle frasi. Presenta una struttura del tipo [0 0 0 0 0 1 1 1 1 1 1 0 0 0], dove la prima sequenza di 0 rappresenta la prima frase, la sequenza di 1 rappresenta la seconda fase mentre gli 0 presenti dopo la sequenza di 1 rappresentano i caratteri di riempimento da non considerare
- Positional Embeddings: array che rappresenta la posizione di un item all'interno del contesto.

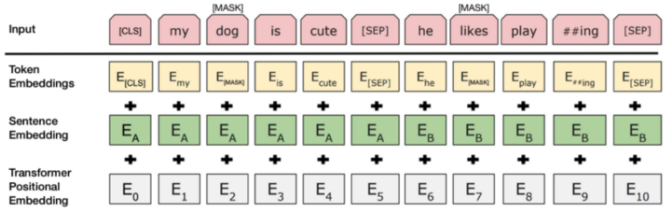


Fig. 3: Rappresentazione dell'input di BERT [3]

Otterremo dunque il seguente risultato come schematizzato in Figura 3:

- Token Embeddings: [101 35888 10264 102 35888 10264 11061 12829 117 12732 16937 102 0 0]
- Segment Embeddings: [0 0 0 0 0 1 1 1 1 1 1 1 0 0]
- Positional Embeddings: [0 1 2 3 4 5 6 7 8 9 10 11 0 0]

V. Metriche di valutazione

Data una query ed un listato di risultati da essa restituiti, l'obiettivo del modello è quello di classificare correttamente i prodotti seguendo la scala ESCI. É facile intuire che si tratta di un problema di classificazzione multiclasse. L'input del modello sarà dunque una coppia ¡query, prodotto¿ dove il prodotto è costituito dai vari attributi descritti nella Sezione II. Abbiamo utilizzato l'F1 Score per valutare le performance. In particolare modo, in quanto le quattro classi risultano sbilanciate (65.20% E, 21.91% S, 2.89% C, 10.00% I), si è deciso di usare una metrica più robusta ovvero la versione micro-average dell'F1 score, come segue:

$$Precision_{\mu\text{-}avg} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FP_i} \tag{1}$$

$$Recall_{\mu\text{-}avg} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FN_i} \tag{2}$$

$$F1\ Score_{\mu\text{-}avg} = 2 \cdot \frac{Precision_{\mu\text{-}avg} \cdot Recall_{\mu\text{-}avg}}{Precision_{\mu\text{-}avg} + Recall_{\mu\text{-}avg}} \tag{3}$$

VI. Architettura

A. BERT

BERT [3] è un modello pre-addestrato con architettura Transformer [4] per task di NLP inizialmente creato per essere integrato all'interno dei motori di ricerca di Google. BERT è stato addestrato su una grande quantità di testo non annotato con due obiettivi diversi: la modellazione del linguaggio mascherato (MLM) e la previsione della frase

successiva (NSP). L'obiettivo MLM è quello di prevedere la parola mascherata in base al contesto, mentre l'obiettivo NSP è quello di prevedere se due frasi sono consecutive o meno.

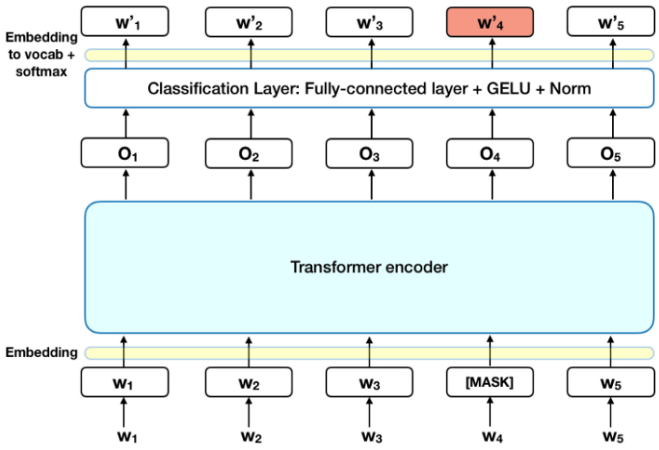


Fig. 4: Architettura di BERT [3]

Il modello BERT è composto da 12 blocchi Transformer [4] come mostrato in Figura 4. La prima keyword è il token `[CLS]` mentre il token di separazione nonchè keyword di chiusura dell'input è `[SEP]`. Il token `[CLS]` è usato per compiti di classificazione, mentre il token `[SEP]` è usato per separare due frasi e per la terminazione dell'input. Può essere anche utilizzato il token `[MASK]` che serve ad indicare una porzione di testo da completare in base al contesto, utilizzabile solo se l'input non inizia con il token di classificazione `[CLS]`.

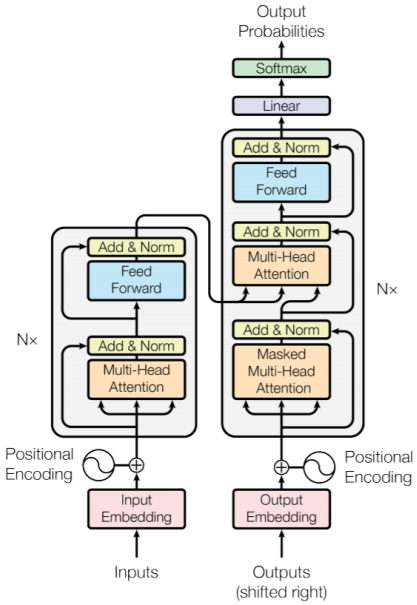


Fig. 5: Architettura del modello Transformer [4]

1) Transformer: Il Transformer [4] è un'architettura all'avanguardia per molti compiti di NLP [3, 13]. Rappresentata in figura 5, è composta da sei componenti: (1) un layer di embedding per la rappresentazione delle parole, (2) l'encoder per la codifica del testo in ingresso, (3) il decoder per la decodifica del testo, (4) un multi-head attention layer, (5) un layer position-wise fully connected e (6) un layer softmax per la predizione.

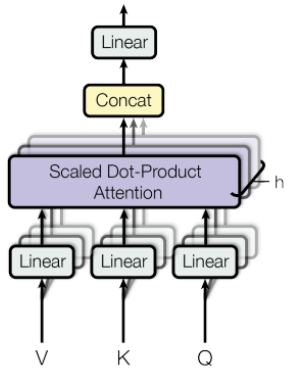


Fig. 6: Multi-Head Attention di dimensione h.

a) Embedding Layer: Il layer di embedding è responsabile della codifica di un token in un vettore di parole. Nell'architettura Transformer, il layer di embedding consiste in due matrici di embedding: E_w in $\mathbb{R}^{d_e \times |\mathcal{V}|}$ e E_p in $\mathbb{R}^{d_p \times |\mathcal{V}|}$, dove $|\mathcal{V}|$ è la dimensione del vocabolario, d_e è la dimensione dell'array di embedding e d_p è la dimensione dell'array posizionale.

Il word embedding è inizializzato come un vettore one-hot e il position embedding è definito come una funzione sinusoidale, così definita:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{\text{model}}}\right)$$

dove pos è la posizione della parola nella frase ed i la dimensione. L'embedding delle parole e l'embedding posizionale vengono sommati e normalizzati. Quindi, l'embedding finale viene immesso nel modello Transformer come input.

b) Multi-head attention layer: in questa architettura, mostrata in figura 6, la funzione di attenzione è espressa come:

$$\hat{x}_i = \frac{\exp\left(\frac{1}{\sqrt{d_k}} q_i^\top W_k k_i\right)}{\sum_{j=1}^{N} \exp\left(\frac{1}{\sqrt{d_k}} q_i^\top W_k k_j\right)} v_i \tag{4}$$

dove $W_k \in \mathbb{R}^{d \times d}$ è una matrice di pesi addestrabile e d_k è la dimensione delle chiavi. Utilizziamo il meccanismo di attenzione a più teste [4] per cercare di evidenziare le correlazioni tra i diversi sottospazi di rappresentazione. Il multi-head attention layer può essere espresso come:

$$\hat{x}_i = \text{Concat}\big(\hat{x}_i^{(1)}, \hat{x}_i^{(2)}, \ldots, \hat{x}_i^{(H)}\big)\mathbf{W}_o, \tag{5}$$

dove $\hat{x}_i^{(h)}$ è la h-esima testa di attenzione per l'i-esimo frame, $W_o \in \mathbb{R}^{d \times d}$ è una matrice di peso addestrabile.

Il multi-head attention layer può essere utilizzato per sostituire l'architettura ricorrente ($e.g$, LSTM) nelle RNN, migliorando così l'efficienza dell'addestramento, ed è altrettanto efficiente nel sostituire i layer fully-connected nei modelli CNN.

c) Encoder: L'encoder è composto da uno stack di N strati identici. Ogni strato è composto da due sottolivelli: un multi-head attention layer e un layer feed-forward posizionale. Nel multi-head attention layer, i vettori delle parole in ingresso vengono proiettati nei vettori query, key e value. Questi vettori vengono poi moltiplicati per le rispettive matrici W^Q, W^K e W^V, e utilizzati nella funzione softmax per calcolare la probabilità di attenzione, così come segue

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Quindi, il vettore di query viene moltiplicato per la probabilità di attenzione e aggiunto al vettore dei value. Infine è presente lo strato feed-forward position-wise, costituito da due layer lineari con una funzione di attivazione ReLU.

d) Decoder: Il decoder è costituito anch'esso da uno stack di N strati identici. Ogni strato è composto da tre sottolivelli: un masked multi-head attention layer, un multi-head attention layer e un layer feed-forward posizionale. Il masked multi-head attention layer viene utilizzato per impedire al modello di essere influenzato dal contesto calcolato in seguito. Essendo l'architettura ricorsiva ad ogni step t il masked multi-head attention rende visibili al modello le parole note fino alla posizione t in modo da forzare la previsione della parola in posizione $t+1$ che all'iterazione successiva sarà nota. Nel masked multi-head attention layer, i vettori key e query vengono moltiplicati per le rispettive matrici dei pesi W^Q e W^K, e vengono passati alla funzione softmax per calcolare la probabilità di attenzione. Il vettore query viene moltiplicato per la probabilità di attenzione e aggiunto al vettore dei value. L'altro multi-head attention layer viene utilizzato per prestare attenzione all'uscita del codificatore. Lo strato di feed-forward posizionale è anch'esso costituito da due layer lineari con una funzione di attivazione ReLU.

e) Softmax: La funzione softmax è una funzione che mappa un vettore di valore reale in una distribuzione di probabilità, cioè tutte le componenti del vettore di uscita sono positive e la somma di tutte le componenti è 1.

La funzione softmax è formalmente definita come segue:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{sum_{j=1}^{n}\exp(x_j)}$$

dove n è il numero di componenti del vettore x.

Nel contesto delle reti neurali, la funzione softmax è tipicamente utilizzata come funzione di attivazione dell'ultimo strato di una rete, quando la rete è utilizzata per la classificazione. L'uscita della rete viene quindi interpretata come una distribuzione di probabilità sulle classi e la classe con la probabilità più alta viene predetta come uscita della rete.

VII. Esperimento

A. QuProBERT

Come spiegato in sezione VI, allo stato dell'arte le migliori architetture per i task di NLP fanno utilizzo dell'architettura Transformer [4]; tra le varie architetture presenti in letteratura, BERT [3] risulta garantire i migliori risultati con minori tempi di addestramento.

Con le seguenti premesse abbiamo sviluppato un nuovo modello basato su quest'ultimo.

Per sviluppare QuProBERT sono state apportate diverse modifiche all'architettura base (BERT Base Uncased Multilanguage) al fine di ottenere un modello specializzato. Nella prima versione di QuProBERT erano stati previsti due layer LSTM, che ben si prestano a task di classificazione su sequenze per loro stessa concezione, ma abbiamo notato che l'utilizzo di questi portava a diverse problematiche:

- **Exploding Gradient**: si verifica spesso in reti neurali profonde quando si tenta di addestrare un modello. Il problema si verifica quando i gradienti della loss function vengono moltiplicati troppe volte durante la backpropagation, causando la loro esplosione e rendendo il modello inutilizzabile.
- **Problemi di parallelizzazione**: per loro struttura le reti ricorrenti eseguono calcoli in sequenza, ciò non permette di poter sfruttare al meglio le potenzialità delle GPUs causando lunghi tempi di addestramento.

Il problema dell'Exploding Gradient è stato risolto utilizzando un layer di normalizzazione così come ideato da Vaswani et al. [4]. Per ogni sequenza rappresentativa di una frase viene calcolata la media e la varianza al fine di normalizzare la sequenza rendendo la varianza ≈ 1 e la media ≈ 0.

Tuttavia il problema della parallelizzazione non poteva essere risolto nell'attuale implementazione delle LSTM in tensorflow. Di conseguenza si è deciso di implementare una rete feed-forward fully connected al fine di sfruttare al massimo le capacità delle GPUs e ridurre drasticamente i tempi di allenamento.

Si è optato dunque per due layer fully connected configurati come descritto in tabella IV.

1) Grid Search: É stata dunque necessaria una lunga fase di grid search al fine di ottenere il miglior modello possibile, si è deciso di variare i soli learning-rate e batch-size come consiugliato da Devlin et al. [3].

	Dense #1	**Dense #2**
Units	128	128
Activation Function	ReLU	ReLU
Dropout	0.5	0.5
Kernel regularizer	L1=1e-5	L2=1e-4
Bias Regularizer	L2=1e-4	L2=1e-4
Activity Regularizer	n/a	n/a
Kernel constraint	MinMax(0-1)	MinMax(0-1)
Bias constraint	n/a	n/a

Tabella IV: Configurazione dei layer fully-connected

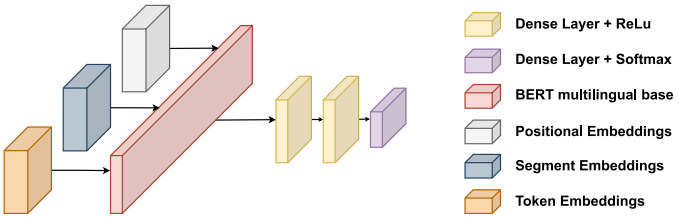


Fig. 7: QuProBERT, Query-Product BERT

2) Pesi per classe: Tenendo conto del forte sbilanciamento presente all'interno del dataset si è deciso di utilizzare dei pesi diversi per ogni classe in modo da penalizzare maggiormente un errore su una classe sottorappresenta, tramite la seguente formula:

$$C_i = \frac{1}{n_i} \cdot \frac{N}{C_{tot}}$$

dove C_i è l'i-esima classe, n_i è il numero di tuple della i-esima classe, N è il numero totale delle tuple del dataset e C_{tot} è il numero di classe totali in cui il dataset è diviso. Così facendo sono stati ottenuti i pesi specificati in tabella V.

Class	Distribution	Weight
Exact	65.21	0.383
Substitute	21.89	1.142
Complement	2.91	8.601
Irrilevant	10.00	2.501

Tabella V: Configurazione dei pesi

VIII. Risultati

I punteggi presentati in tabella VI mostrano i risultati finali ottenuti dai modelli al variare degli iperparametri specificati in tabella III, valutati secondo il Micro F1-score.

Possiamo a partire dai risultati qui presentati trarre le seguenti conclusioni:

- **Learning Rate**: il learning rate elevato, $5e-5$ / $4e-5$, come utilizzato nei modelli M1-4 ha fatto convergere il modello troppo rapidamente verso una soluzione non ottimale. Nei modelli M7-8, dove è stato utilizzato un learning rate di 10^{-5}, essi risultano convergere troppo lentamente verso il minimo, molte più epoche risulterebbero necessarie per arrivare ad un punto di minimo. Risulta evidente che il learning rate intermedio di $3e-5$ ha fornito le migliori performance facendo convergere rapidamente il modello verso un possibile punto di minimo.

Learning rate(Adam)	Batch Size	Micro F1-score
$5e-5$	16	0.717
$5e-5$	32	0.688
$4e-5$	16	0.686
$4e-5$	32	0.679
$3e-5$	**16**	**0.748**
$3e-5$	32	0.723
$1e-5$	16	0.613
$1e-5$	32	0.623

Tabella VI: Punteggi Micro F1 ottenuti

- **Batch size**: andando a considerare esclusivamente la batch size (32/16) dei modelli che hanno ottenuto le migliori prestazioni (M5-6) possiamo concludere, non sorprendetemente, che una batch size troppo elevata ha portato ad una scarsa generalizzazione.
- **Dropout**: come è ben noto, i modelli che utilizzano un MLP come classificatore beneficiano dell'applicazione della regolarizzazione tramite Dropout. Nel modello proposto ad entrambi i layer fully-connected è stato applicato un dropout con probabilità del 50% per ottenere la massima varianza della distribuzione.
- **Regolarizzazione**: la regolarizzazione si è rivelata essere, non sorprendentemente, di fondamentale importanza in un modello complesso come quello costruito, e la configurazione ottimale sembra essere il porre un termine di regolarizzazione sia sui bias e sui pesi dei layer densi interni, ma non sui valori di attivazione. Applicando la regolarizzazione $L1 = 10^{-5}$ si riduce a zero il coefficiente delle caratteristiche meno importanti, eliminando così alcune caratteristiche, motivo per cui è stata applicata solo al primo layer denso. Invece la regolarizzazione $L2 = 10^{-4}$ è stata applicata per cercare di impedire al modello di andare in overfitting.

Andando adesso ad analizzare la tabella comparativa VII, possiamo effettuare ulteriori osservazioni. L'architettura QuProBert, in figura 7, risulta surclassare la baseline proposta da Amazon [10].

Inoltre avendo lo stesso modello proposto da Amazon, modificato del solo layer finale di output, ottenuto performance paragonabili a quelle presenti in letteratura sulla classificazione binaria dei prodotti (Rilevante - Non rilevante) possiamo concludere con buon grado di sicurezza che sul medesimo task il modello QuProBert sarebbe capace di ottenere performance pari se non superiori alle altre qui presentate.

IX. Conclusioni

In questo lavoro abbiamo analizzato il dataset fornito nel corso della KDD'22 Cup in modo tale da renderlo utilizzabile per la corretta risoluzione del problema che ci è stato proposto. Inoltre, abbiamo anche effettuato un'analisi dettagliata di come poter processare le varie query in diverse lingue cercando di rendere il nostro sitema il più immune possibile dai principali problemi legati alla differenze linguistiche.

La nostra analisi ha anche delineato le principali questioni che dovrebbero essere affrontate nella ricerca futura in questo settore. In particolare, è necessario applicare un massiccio processo di tokenizzazione per poter uniformare tutti i dati, in modo tale da facilitare l'estrazione semantica del contesto e allo stesso tempo essere conformi agli standard esistenti per quanto riguarda task di NLP.

In futuro, sono necessarie ulteriori ricerche per migliorare l'integrazione di dati tra le varie lingue, come ad esempio sfruttare le conoscenze linguistiche per applicare delle tecniche di 'pruning' in un contesto semantico.

Infine pensiamo che il modello proposto possa essere considerato come un buon punto di partenza per i motori di ricerca

Model name	Binary Classification	ESCI Classification	ESCI English	ESCI Spanish	ESCI Japanese
BERT-base (eBay Inc. baseline)	0.822	-	-	-	-
eBERT-base (eBay Inc.)	0.870	-	-	-	-
BERT-student (eBay Inc.)	0.861	-	-	-	-
BiLSTM (eBay Inc.)	0.858	-	-	-	-
BertBiLSTM (eBay Inc.)	0.863	-	-	-	-
S-BiLSTM (eBay Inc.)	0.852	-	-	-	-
Frozen BERT MLP (Amazon baseline)	0.852	0.655	0.685	0.580	0.595
QuProBert (Amazon KDD)	-	0.748	0.776	0.683	0.694

Tabella VII: Comparativa dei vari modelli presenti in letteratura e dell'architettura proposta

basati sulla semantica che, a nostro avviso, saranno in grado di migliorare l'esperienza dell'utente garantendo risultati molto più pertinenti alla richiesta.

REFERENCES

[1] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202.

[2] Daniel Cer et al. "Universal Sentence Encoder for English". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 169–174. DOI: 10.18653/v1/D18-2029. URL: https://aclanthology.org/D18-2029.

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[4] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[5] Alireza Bagheri Garakani et al. "Improving Relevance Quality in Product Search using High-Precision Query-Product Semantic Similarity". In: *Proceedings of the Fifth Workshop on e-Commerce and NLP (ECNLP 5)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 44–48. DOI: 10.18653/v1/2022.ecnlp-1.6. URL: https://aclanthology.org/2022.ecnlp-1.6.

[6] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.

[7] Leonard Dahlmann and Tomer Lancewicki. "Deploying a BERT-based Query-Title Relevance Classifier in a Production System: a View from the Trenches". In: *CoRR* abs/2108.10197 (2021). arXiv: 2108.10197. URL: https://arxiv.org/abs/2108.10197.

[8] Matej Kocián et al. "Siamese BERT-based Model for Web Search Relevance Ranking Evaluated on a New Czech Dataset". In: *CoRR* abs/2112.01810 (2021). arXiv: 2112.01810. URL: https://arxiv.org/abs/2112.01810.

[9] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *CoRR* abs/2003.10555 (2020). arXiv: 2003.10555. URL: https://arxiv.org/abs/2003.10555.

[10] Chandan K. Reddy et al. *Shopping Queries Dataset: A Large-Scale ESCI Benchmark for Improving Product Search*. 2022. DOI: 10.48550/ARXIV.2206.06588. URL: https://arxiv.org/abs/2206.06588.

[11] Leonard Richardson. *Beautiful Soup - HTML Parser*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[12] HuggingFace. *HuggingFace - Summarization Task*. Ed. by HuggingFace. 2021. URL: https://huggingface.co/docs/transformers/tasks/summarization.

[13] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).