Università degli studi di Torino

Dipartimento di Informatica

Tesi di Laurea Magistrale in Informatica

Ricerca ibrida con vector database: Sperimentazione e realizzazione POC

Relatore:
Prof. Daniele Radicioni
Correlatore:
Dott. Davide Colla

Candidato:
Alberto Campini
M: 885128

Anno Accademico 2022/2023

26 Ottobre 2023

Dichiarazione di originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Sommario

La rivoluzione portata avanti negli ultimi anni dell'Intelligenza Artificiale ha semplificato il trattamento di alcuni task, producendo al contempo una serie di nuove sfide, fra cui la rappresentazione e la gestione dei dati. Questa tesi, dopo aver introdotto le possibili forme di rappresentazione vettoriale dell'informazione, propone una trattazione dettagliata del funzionamento e dell'architettura dei vector database. Nel capitolo centrale è introdotta la fase sperimentale, finalizzata a realizzare la ricerca ibrida al fine di migliorare l'accuratezza del processo di reperimento dell'informazione. Questo lavoro è affrontato sia dal punto di vista teorico sia da quello pratico, utilizzando diversi approcci e sfruttando il database vettoriale Qdrant. La trattazione si conclude con la presentazione di un proof of concept che utilizza le potenzialità descritte dei vector database in un contesto applicativo industriale reale, sviluppato per l'azienda Cluster Reply.

Ad Alberto
Piero,
Daniela
e Giulia

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Questi ultimi anni sono stati caratterizzati da uno sviluppo senza precedenti nell'industria dell'intelligenza artificiale e possiamo dire di trovarci nel mezzo della cosiddetta AI[1] revolution. Le grandi aziende investono moltissime risorse in ricerca e innovazione in questo ambito e non mancano di certo le sfide che si pongono nel percorso di sviluppo. La costruzione di LLMs[2] sempre più complessi e con numero di paramenti interni incrementali rendono cruciale riuscire a gestire al meglio la gestione dei dati e la loro elaborazione.

I dati come in tutti gli ambiti dell'informatica moderna anche all'interno del panorama dell'intelligenza artificiale sono parte integrante del processo di costruzione dei modelli stessi, infatti vengono usati in grandissime quantità per andare, in prima battuta, ad addestrare il modello e in un secondo momento a fare un fine-tuning su un ambito specifico.

Le informazioni, spesso, hanno una rappresentazione particolare, il testo viene sostituito da numeri avvicinandosi molto di più ad una rappresentazione comoda ed efficiente per le macchine piuttosto che per gli esseri umani; cercando di andare a rappresentare concetti latenti dietro intere frasi o insiemi di documenti.

Uno dei primi obiettivi che si pone questo lavoro è quello di andare a trattare nel dettaglio quali sono e come sono nate le differenti rappresentazioni dell'informazione vettoriale, analizzandone le differenti caratteristiche, punti di forza e limiti. La trattazione dal punto di vista teorico sarà necessaria per affrontare le sezioni successive in cui ci si sposta verso un approccio pratico sperimentale degli argomenti trattati.

Come accennato, la rappresentazione non è l'unica sfida a cui la comunità scientifica è stata portata ad affrontare, ma vi è anche la gestione di questi dati,

[1] Intelligenza Artificiale

[2] Large Language Models

che, come vedremo assumono la forma di un vettore numerico. Da questa necessità e dal successo dei database noSQL nascono i vector database, sistemi che facilitano la gestione di grandi quantità di dati vettoriali. Questa tecnologia verrà trattata nel dettaglio andando a capire quali sono le caratteristiche architetturali che distinguono i vector database da approcci più comuni, sino alla gestione dell'informazione e come possono essere sfruttati in casi d'uso reali.

La trattazione, dopo aver introdotto background teorico necessario degli elementi utili, proseguirà andando ad affrontare dal punto di vista sperimentale l'hybrid search, particolare approccio per fondere rappresentazioni differiti, con il fine ultimo di migliorare il processo di retrieval dei database vettoriali. In primo luogo si cercherà di dare una definizione e successivamente esplorare quali sono le tecniche utilizzate in ambiti differenti da sfruttare come base per tentare approcci differenti volti a migliorare le performance.

L'utilizzo del database vettoriale open source Qdrant accompagnerà tutta la parte sperimentale della trattazione e verrà utilizzato come parte centrale della trattazione dei vari approcci alla tecnica di retrieval volta a sfruttare rappresentazioni sparse e dense dell'informazione.

Il lavoro terminerà con l'analisi e presentazione di un proof of concept sviluppato per Cluster Reply, azienda in cui lavoro da 2 anni a questa parte. Quest'ultima sezione ha l'obiettivo di far toccare con mano come le tematiche trattate in questa tesi possono essere declinate in ambiti reali e, non per forza legati a tali tecnologie. Dopo aver dato il contesto necessario nel quale è stato portato avanti lo sviluppo per capire quale fosse l'obiettivo ultimo della proposta fatta al cliente, si affronterà tutto lo sviluppo della soluzione focalizzandosi principalmente sugli elementi trattati in questo lavoro.

In questa tesi dunque verranno affrontati i differenti tipi di rappresentazione dell'informazione e appoggiandosi alla sperimentazione della ricerca ibrida sul database vettoriale Qdrant si cercherà di portare alla luce aspetti teorici rilevanti. andando a concludere con la presentazione di un caso d'uso proveniente dal modo lavorativo reale dando una panoramica a 360° degli argomenti affrontati.

Capitolo 2

Related Work

Il questo capitolo verranno introdotti tutti gli elementi teorici necessari per capire e trattare dal punto di vista critico il lavoro sperimentale svolto in questa tesi e presentato nel capitolo successivo. La trattazione si pone l'obiettivo di gettare le basi teoriche per capire i differenti approcci alla rappresentazione vettoriale dell'informazione e quali sono state le esigenze ed intuizioni storiche che hanno portato fino allo stato dell'arte odierno.

In questa sezione verranno inoltre presentati e approfonditi tecnicamente i database vettoriali per introdurre le caratteristiche e potenzialità, saranno un punto centrale sia della parte sperimentale che del proof of concept presentato. Come esempio di architettura e funzionamento verrà dettagliato il Database Vettoriale Open Source Qdrand[1].

[1] qdrant.tech

2.1 Rappresentazione dell'informazione

Oggigiorno non si può trattare l'informatica senza parlare di dati e della loro rappresentazione, per riuscire a capire come siamo arrivati ai database vettoriali e rappresentazioni complesse come gli embeddings è bene fare un passo indietro cercando di capire e approfondire quali siano state le esigenze che hanno portato la realizzazione di differenti approcci alla rappresentazione dell'informazione.

Come primo passo è bene capire che cosa intendiamo con "informazione da rappresentare", in questo ambito i nostri dati sono generalmente del testo strutturato, che può essere di differenti dimensioni ed organizzato in collezioni di documenti. Generalmente definiamo con corpus un insieme di documenti strutturati dal quale volgiamo ottenere informazioni oppure sfruttare per qualche task specifico.

Volendo lavorare e trattare questi insieme di dati sfruttando meccanismi automatici più o meno complessi ci scontriamo subito sul come rendere tali documenti utilizzabili da algoritmi di apprendimento automatico o da reti neurali profonde. I dati sotto forma di testo sono facilmente fruibili da parte nostra, ma più complessi da gestire in modo sistematico e standardizzato dalle macchine.

Le grosse quantità di testo spesso infatti vengono utilizzate come punto di partenza per andare ad eseguire specifici task. Prima di approfondire i passi fatti per ottimizzare la rappresentazione dell'informazione è importante capire quali possono essere i task principali di cui parliamo che sfruttano corpus testuali dando quindi un'idea di dove vogliamo arrivare.

Classificazione

Il task della classificazione in NLP[2] è una delle principali attività nell'ambito dell'elaborazione del linguaggio naturale. In realtà il task della classificazione possiede una vasta gamma di sotto task in cui, ognuno dei quali richiede una strategia specifica per l'elaborazione del testo e la costruzione del modello.

Quello però che li accomuna è il tentativo di far prevedere al modello dato un training set delle etichette per insieme di dati mai visti. Alcuni esempi di task di classificazione in NLP sono:

- Sentiment Analysis: l'obbiettivo in questo task è quello di assegnare un'etichetta di categoria (positiva, negativa o neutra) a una porzione di testo, solitamente una recensione o un commento. Generalmente viene realizzato un modello in grado di classificare le recensione da parte dei clienti su specifici

[2]Natural Language Process

prodotti in vendita così da poter ottenere metriche di valutazioni sull'indice di gradimento oltre alle classiche stelline.

- Classificazione di Intenti Utente: Con l'avvento e la diffusione dei chatbot in differenti ambiti l'esigenza di riuscire a capire l'intento dell'utente che vi sta interagendo è diventata centrale. Viene in soccorso anche in questo ambito specifico la classificazione, che permette di identificare dato un messaggio utente a quale macro categoria si sta riferendo e di conseguenza permettere al chatbot di rispondere in maniera pertinente.

- Classificazione di Categorie o topic modelling: Dato un testo o insieme di testi l'obiettivo è quello di predire in automatico una categoria di riferimento o topic del documento stesso. Questo meccanismo permette di descrivere e categorizzare documenti in modo automatico.

- Rilevamento di Linguaggio Ostile o Spam: Questo task e polivalente e si cala in moltissimi ambiti, dall'analisi della casella postale per predire se una mail è spam o meno, ma anche nella moderazione online andando ad oscurare messaggi con contenuti inappropriati per gli altri utenti. Il modello in questo caso può essere più o meno complesso, infatti possiamo trovare classificatori semplici basati su pattern matching ed etichette sino a sistemi complessi che sfruttano reti neurali profonde.

- Rilevamento di Bug o Problemi nei Testi Tecnici: Nell'ambito dello sviluppo software, è possibile classificare i problemi segnalati dagli utenti in base alla loro gravità o al componente del software interessato, così da poter classificare e prioritizzare gli incident.

La soluzione per tutti i task di classificazione elencati generalmente coinvolge la raccolta di dati etichettati (testi con etichette di classe inserite manualmente da persone) per addestrare un modello di apprendimento automatico. Successivamente, il modello viene valutato su dati di test per misurare la sua capacità di generalizzazione.

La scelta della strategia del modello dipenderà dal task specifico e dalla quantità e qualità di dati disponibili per fare l'addestramento.

Information retrieval

Uno dei task base della NLP è l'information retrieval, termine coniato da Calvin Mooers [1] alla fine degli anni quaranta del Novecento ed oggi è usato quasi esclusivamente in ambito informatico.

L'obiettivo del task è quello di garantire all'utente, in seguito ad una sua ricerca, i documenti e le informazioni che rispondono alla sua richiesta. Per rendere possibile

ciò è importante capire come rappresentare i dati, la query e come confrontarle per individuare l'informazione corretta.

La trattazione successiva verterà esattamente su queste tematiche andando ad analizzare differente tecniche di rappresentazione vettoriale e metriche di similarità.

Summarization

Come suggerisce il nome del task l'obiettivo è quello dato un documento o un insieme di documenti di ottenere un riassunto che mantenga coerente il senso di partenza, Questo task può essere fatto principalmente in due modi:

- Estrattivo: viene scelto un compression rate e sulla base di quello si sceglie cosa tenere e cosa eliminare simulando di fatto il processo di sottolineatura di un testo. Le parti importanti faranno parte del risultato finale, mentre quelle meno informative verranno escluse. Naturalmente questo è un approccio semplicistico dato che non il modello non riesce a capire effettivamente il senso del testo, ma semplicemente riordina le sue parti in base all'importanza.

- Astrattivo: Il livello di complessità aumenta rispetto alla versione estrattiva, infatti un riassunto astrattivo, rielabora il concetto di fondo del documento con un minor numero di parole di conseguenza non è sufficiente andare ad analizzare il corpus, ma deve essere datato di un modello generativo per andare a realizzare il riassunto stesso. Spesso questa tipologia di task è demandata ai Large Language Model date le loro capacità di "comprensione" e di generazione di testo.

Per riuscire ad elaborare e trattare grosse quantità di dati in modo automatico andando di fatto a rendere possibili i task appena citati serve una rappresentazione numerica, quindi più facilmente gestibile dalle macchine e che sperabilmente approssimi nel miglior modo possibile quella testuale effettivamente contenuta all'interno dei documenti stessi.

2.1.1 Vettori sparsi

I modelli vettoriali o distribuzionali sono generalmente basati su una matrice di co-occorrenza, essa è un buon modo di rappresentare quanto frequentemente le parole si verificano insieme cercando di modellare significato sulla base di queste relazioni. Verranno prese in esame rappresentazioni vettoriali per parole e per documenti all'interno di una collezione.

Parole come vettori

Il primo modo utilizzato per rappresentare dati testuali sotto forma numerica sfrutta un approccio concettualmente semplicistico conosciuto come one-hot encoding. Questa metodologia storicamente nasce per sopperire all'esigenza di rendere trattabili dati categorici generici all'interno di sistemi automatici andando a trasformare le categorie in vettori binari. questo tipo di rappresentazione garantisce che non ci sia alcun ordine o gerarchia intrinseca tra le categorie codificate, evitando quindi di introdurre erroneamente relazioni ordinali tra di esse.

Facendo un esempio tangibile per cogliere l'intuizione alla base di questo metodo, se si volesse modellare la categoria "rosso", all'interno del seguente dizionario verde, rosso, giallo, blu il vettore sarebbe definito come:

$$(0,1,0,0) \tag{2.1}$$

Se invece volessimo rappresentare il blu il vettore target sarà:

$$(0,0,0,1) \tag{2.2}$$

In letteratura esiste una variante speculare di quanto detto, definito come one-cold encoding, intuitivamente è la medesima rappresentazione appena enunciata in cui però la categoria target sarà a 0 mentre tutti gli altri saranno ad 1.

Rendere utilizzabile il one-hot con insieme di testo è dunque immediato, infatti andando a trattare i termini come categorie riusciamo a modellare i termini. Infatti permette di rappresentare uno specifico termine all'interno di un documento come vettore numerico sparso che ha come cardinalità il numero di parole nel dizionario target in cui la posizione che coincide con la parola che volgiamo modellare sarà valorizzata come 1 mentre tutte le altre a 0.

Con questo approccio è dunque possibile rappresentare parole come vettori numerici e adottando la stessa idea è immediato rappresentare insieme di parole come matrici in cui ogni riga rappresenta il singolo termine all'interno del testo. Dunque la rappresentazione è passata da un insieme denso di termini ad un insieme di vettori numerici sparsi che colgono l'occorrenza di un termine specifico all'interno della frase. Questo è stato un passo importante che ha aperto la strada per una rappresentazione sfruttabile dai sistemi automatici in modo efficiente.

Ragionando sulla rappresentazione vettoriale proposta sino ad ora è lampante una mancanza importante, andando a modellare ogni singolo termine come un one-hot vector andiamo di fatto a ripetere un vettore uguale per ogni occorrenza del termine all'interno del testo riducendo di molto l'efficienza e aumentando la dimensionalità della rappresentazione.

Esistono tuttavia altre possibilità per rappresentare come vettore una parola, non più come vettore binario, ma bensì come term-context matrix o conosciuta

anche word-word matrix. Come suggerisce il nome del meccanismo una parola viene rappresentata dalla co-occorrenza del termine stesso in relazione ad altri termini, infatti ogni parola è caratterizzata da un vettore in cui ogni dimensione rappresenta un possibile termine che appartiene o meno al suo contesto. Più in generale per un insieme di termini avremo una matrice di dimensioni $|V| \times |V|$ in cui ogni cella rappresenta il numero di co-occorrenze tra il termine in riga (target) e quello di colonna (contesto).

Il concetto di contesto può essere gestito in molti modi, un esempio potrebbe essere di andare a contare parole che appaiono nello stesso documento. Tuttavia la ricerca è giunta a valutare come un approccio più preciso l'analisi del contesto intra-documento, infatti è troppo dispersivo valutare l'intero documento come contesto di una parola e inoltre tende ad essere soggetto a alterazioni in base alla lunghezza del documento stesso.

La tecnica che permette dia vere migliori prestazioni è quella di valutare il contesto di una parola target all'interno di una slinding window posta attorno alla parola che stiamo cercando di andare a valorizzare. Questo permette di valutare come contesto solo termini che co-occorno "vicini" al termine di nostro interesse standardizzando di fatto il concetto di contesto e rendendo la metrica più robusta.

Utilizzando una finestra di 4 a sinistra e 4 a destra attorno alla nostra parola target su questo piccolo corpus esemplificativo è possibile costruire la rappresentazione word-word matrix come in tabella 2.1.

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Tabella 2.1: Vettori di co-occorrenza per quattro parole nel corpus di Wikipedia, mostrando sei delle dimensioni (scelte appositamente per esemplificazione). Si noti che un vettore reale avrebbe molte più dimensioni e quindi sarebbe molto più sparso. Esempio tratto da [2]

Dalla tabelle delle co-occorenze è possibile notare che i termini "cherry" e "strawberry" sono più simili rispetto agli altri (sia "pie" che "sugar" appaiono nel

loro contesto), mentre termini come "digital" e "information" sono molto differenti rispetto ai due ma simili tra loro. Andando ad inserire in uno spazio bi-dimensionale i valori per le features "computer" e "data" notiamo questa similarità

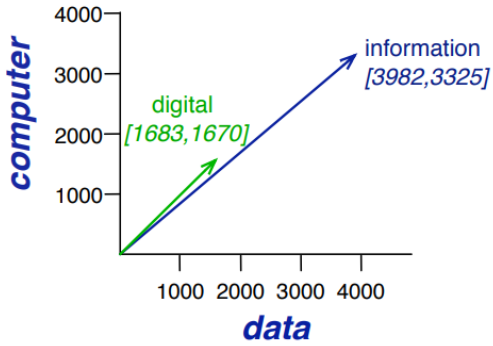

Figura 2.1: Una visualizzazione spaziale dei vettori di parole per "digital" e "information", mostrando solo due delle dimensioni, corrispondenti alle parole "data" e "computer". Esempio tratto da [2]

Da questo esempio è possibile notare come sia tangibile la teoria che ha guidato la rappresentazione vettoriale delle parole, cioè che *termini che accadono in contesti simili, possiedono significati simili*[3]

[3]Harris, Z. S. (1954)

Documento come vettore

Cambiando invece approccio ma sfruttando una rappresentazione sempre sparsa, non si rappresenta un termine con un vettore binario, ma gli assegniamo un peso all'interno del documento stesso che ne rappresenta sia l'occorrenza che l'importanza. Adottando questa intuizione è possibile rappresentare un documento come un vettore singolo, che avrà numero di dimensioni pari al numero di termini presenti e ogni valore della feature, detta term-weigth, non sarà altro che un punteggio associato ad un termine che ne rappresenta l'importanza.

In generale dunque un documento d_j all'interno di una collezione sarà rappresentato come:

$$d_j = (w_{1,j}, w_{2,j}, w_{3,j}, ..., w_{n,j}) \tag{2.3}$$

In questa rappresentazione l'elemento $w_{i,j}$ individua l'elemento i fra gli N complessivamente presenti all'interno della collezione, j invece rappresenta il j-esimo documento della collezione. In parole semplici dunque un'intera collezione di documenti sarà rappresentabile come una matrice di documenti, in cui ogni documento avrà tante dimensioni quante il numero di termini che occorrono, questa rappresentazione è definita term-document matrix.

In questa specifica rappresentazione andiamo a cogliere la co-occorenza delle parole nei differenti documenti:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Tabella 2.2: Term-document matrix di 4 parole per 4 opere di Shakespeare. Ogni cella rappresenta l'occorrenza del termine (riga) all'interno del documento (colonna). Esempio tratto da [2]

La matrice termine-documento 2.2 è stata definita come parte del modello di spazio vettoriale per il recupero delle informazioni [3]. All'interno di questa rappresentazione i vettori per i documenti sono facilmente individuabili, infatti ogni colonna rappresenta il vettore del documento di riferimento, quindi la rappresentazione vettoriale di "As You Like It" sarà:

$$d_{AsYouLikeIt} = (1,114,36,20) \tag{2.4}$$

in cui ogni singolo elemento è una dimensione del vettore, e è intuitivo capire che ad esempio per la prima dimensione "battle" i vettori di "As You Like It" e "Twelfth Night" sono simili perché la parola occorre rispettivamente 1 volta e 0 volte.

A questo punto è possibile pensare ai documenti all'interno della matrice 2.2 come dei punti in uno spazio |V| dimensionale, in questo caso specifico lo spazio è 4-dimensionale. Andando a scegliere arbitrariamente 2 features è possibile visualizzare la co-relazione tra documenti differenti. Intuitivamente tanto più l'occorrenza delle parole è simile tanto più i vettori sono simili.

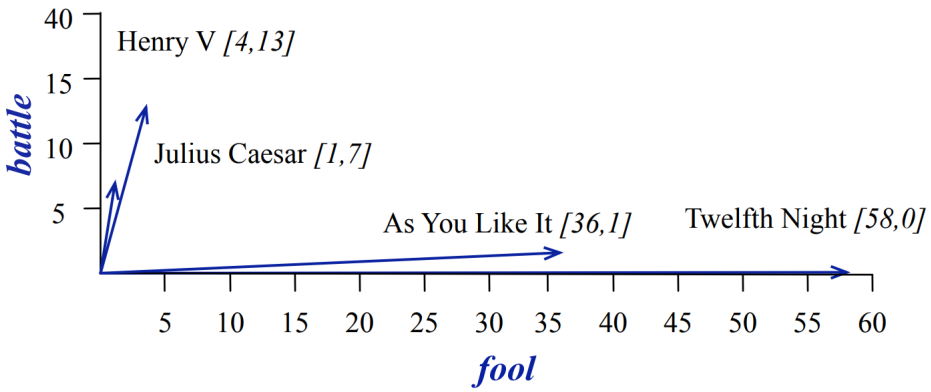

Figura 2.2: Una visualizzazione spaziale dei vettori dei documenti per i quattro documenti delle opere di Shakespeare, mostrando solo due delle dimensioni, corrispondenti alle parole "battle" e "fool". Le commedie hanno valori elevati per la dimensione "fool" e valori bassi per la dimensione "battle". Esempio tratto da [2]

Le matrici termine-documento furono originariamente definite come un modo per trovare documenti simili per il compito di recupero delle informazioni da documenti. Due documenti simili tenderanno ad avere parole simili, e se due documenti hanno parole simili, i loro vettori colonna tenderanno ad essere simili.

I vettori per le commedie "As You Like It" [1, 114, 36, 20] e "Twelfth Night" [0, 80, 58, 15] si assomigliano molto l'uno all'altro, più di quanto assomiglino a "Julius Caesar" [7, 62, 1, 2] o "Henry V" [13, 89, 4, 3]. Questo è evidente dai numeri grezzi; nella prima dimensione (battaglia), le commedie hanno numeri bassi e gli altri hanno numeri alti, e possiamo vederlo visivamente nella Figura 2.2. All'interno della sezione 2.1.3 vedremo come quantificare in modo più formale questa intuizione.

2.1.2 Vettori densi

Fino a questo momento è stata trattata la rappresentazione come vettori lunghi e sparsi con una dimensione corrispondente alle parole nel dizionario o all'interno della collezione presa in analisi, ora introduciamo una rappresentazione molto più efficiente, ma meno intuitiva chiamata embeddings. Essi sono sempre vettori, ma molto più densi e corti, generalmente hanno una dimensionalità che varia tra le 50

fino a 2000 dimensioni, di gran lunga inferiori al numero di termini all'interno del dizionario.

Le dimensioni diversamente da quanto visto non hanno una vera e propria interpretabilità diretta per l'essere umano, quindi non si devono più pensare come una frequenza relativa all'emissione di termini assieme, inoltre come detto sono dense cioè le features posseggono pochissimi zeri e possono assumere valori reali, quindi anche negativi.

Risulta che i vettori densi funzionano meglio in ogni compito di NLP rispetto ai vettori sparsi. Sebbene non comprendiamo completamente tutte le ragioni di ciò, abbiamo delle intuizioni. Rappresentare le parole come vettori densi a 300 dimensioni richiede ai classificatori di apprendere molte meno variabili rispetto a quando rappresentiamo le parole come vettori a 50.000 dimensioni, e questo spazio dei parametri più piccolo potrebbe aiutare nella generalizzazione e nella prevenzione dell'overfitting.

I vettori densi potrebbero anche essere più efficaci nel catturare la sinonimia. Ad esempio, in una rappresentazione con vettori sparsi, le dimensioni per la stessa parola proveniente da due lingue differenti come "car" e "automobile" sono distinte e non correlate; i vettori sparsi potrebbero quindi non riuscire a catturare la somiglianza tra una parola con "car" come vicina e una parola con "automobile". Gli embedding, astraendo la rappresentazione a livello semantico riescono a porre vicini i vettori di "car", "automobile" e "macchina" nonostante siano termini differenti anche provenienti da lingue distinte.

Le features degli embeddigs come già detto non sono direttamente riconducibili a frequenze o numeriche ottenibili analizzando in modo classico un testo ma, in quasi tutte le implementazioni, rappresentano i pesi ottenuti durante la fase di training del modello che ha permesso l'identificazione del contesto della parola target. Generalmente i modelli utilizzati sono reti neurali profonde e gli embeddigs non sono altro che la fotografia dei pesi dell'ultimo strato della rete.

Embeddings statici

L'approccio più semplice e comunque è quello di apprendere embeddigs statici, cioè vettori fissi che rappresentano set di termini. Questi vettori catturano le co-occorrenze tra le parole ed il loro contesto all'interno dei corpus su cui è addestrato il modello, quello che non riescono a fare è di modificare la rappresentazione in base all'utilizzo del termine stesso. Ad esempio l'embegging del termine "bank" conterrà al suo interno sia il senso di bank come istituto finanziario che come riva del fiume.

Uno dei metodi di apprendimento di embeddigs statici più semplice è quello utilizzato da Word2vec che però permette di capire in modo tangibile che cosa rappresentano effettivamente i valori delle features presenti.

L'algoritmo in questione è SGNS[4] e permettere di apprendere un vettore denso a partire da un corpus testuale, per fare ciò semplifica l'approccio utilizzando una semplice logistic regression piuttosto che reti neurali profonde. L'idea è quella di apprendere un modello capace di separare linearmente parole appartenenti al contesto del temine target per cui stiamo calcolando l'embedding da termini lontani che non co-occorno al temine (da qui il nome with negative sampling).

Gli esempi negativi sono cruciali per permettere al modello di generalizzare e non fare overfitting sul dataset con cui è stato fatto il training. Il risultato di questo sarà un modello capace di prevedere se una parola appartiene o meno al contesto del nostro termine che stiamo rappresentando in forma vettoriale. Similmente a come visto per i vettori sparsi che modellano le parole anche in questo caso il contesto è definito da una finestra attorno al termine target gli esempi negativi sono invece parole casuali, spesso contengono del rumore per rendere ancora più robusto il modello.

Gli embeddings oltre che essere più efficienti posseggono delle caratteristiche interessanti, infatti tali vettori contengono al loro interno una sorta di astrazione semantica del significato del termine permettendo di realizzare relazioni e confronti tra vettori. Prendendo il vettore che rappresenta uomo e il vettore che rappresenta re ci accorgiamo che ha le stesse caratteristiche che differenzia il vettore di donna e regina. Andando a sottrarre il vettore di uomo a quello di re e successivamente sommando quello di donna troviamo qualcosa di molto vicino alla rappresentazione di regina. Questo ragionamento si può fare in differenti contesti un altro esempio è quello delle capitali sottraendo il vettore di Francia a quello di Parigi e successivamente sommando quello di Italia otteniamo un vettore denso molto simile a quello di Roma.

Come già introdotto ad inizio di questa sezione gli embeddings statici non permettono di modificare la rappresentazione in base all'uso puntale del termine essendo esposti al meaning confaltion problem, per riuscire ad adempiere all'esigenza di avere vettori differenti in base al contesti bisogna necessariamente ricorrere agli Embeddings contestuali

Embeddings contestuali

Per parlare di embedding contestuali è necessario introdurre il concetto di Large language model. Un LLM è un tipo di sistema di intelligenza artificiale progettato

[4]skip-gram with negative sampling

per comprendere e generare testo naturale in modo coerente e contestualmente appropriato. Questi modelli sono caratterizzati da un addestramento self-supervised su grandi quantità di testo preso da internet e altre fonti per apprendere la struttura della lingua, la grammatica, il contesto e le relazioni semantiche tra le parole.

Architetturalmente i large language model sono basati su reti neurali profonde, come le reti neurali trasformative comunemente conosciuti con il nome di Transformers. Queste architetture sono state fondamentali nel migliorare le capacità di comprensione del linguaggio naturale delle macchine, grazie sopratutto al concetto della self-attenction [4].

Questi modelli sono in grado di comprendere il significato di un testo, rispondere a domande, completare frasi e generare nuovo testo in modo coerente e semanticamente corretto. Possono essere utilizzati per una vasta gamma di compiti, come chatbot, traduzione automatica, riassunti automatici, generazione di testi creativi e altro ancora data la loro capacità

Uno dei punti grandi di forza dei LLM è la loro capacità di considerare il contesto e generare sulla base di esso. Possono comprendere la relazione tra le frasi e rispondere in modo contestualmente appropriato. Questo li rende utili per conversazioni fluide e interazioni che richiedono una comprensione profonda del contesto, vederemo come questa proprietà sarà fondamentale nella trattazione della POC alla sezione 4.

Nonostante le loro capacità impressionanti, i large language model hanno anche limitazioni, sopratutto dovuto dalla loro natura. Possono produrre risposte non corrette o bias se addestrati su dataset di addestramento parziali o contenenti problematiche. Inoltre, possono avere difficoltà con la comprensione di concetti complessi o contesti poco chiari. Lavorando inoltre sul legame tra le parole in base al contesto non effettuano un vero e proprio processo di ragionamento e questo può portare ad incorrere in errori grossolani.

La panoramica è necessaria per capire come si è giunti alla realizzazione degli embedding contestuali, infatti come suggerisce il nome si basano si sui termini presenti, ma anche al loro utilizzo nel contesto presente. Sono una forma avanzata di rappresentazione delle parole in un modello di linguaggio. A differenza degli embedding statici, che assegnano a ciascuna parola un vettore fisso che non tiene conto del contesto, gli embedding contestuali tengono conto del contesto circostante in cui appare una parola in una frase o un documento. Questo rende gli embedding contestuali più ricchi di informazioni e adatti per molti task NLP.

Un esempio di modello per generare Embedding contestuali è BERT[5] sviluppato da Google AI Research presentato per la prima volta nel 2018 [5].

[5]Bidirectional Encoder Representations from Transformers

La caratteristica distintiva di BERT è la sua capacità di comprensione bidirezionale del contesto. A differenza dei modelli precedenti, che consideravano il contesto solo in una direzione (ad esempio, solo le parole precedenti), BERT può considerare simultaneamente sia il contesto a monte che quello a valle di una parola. Questo migliora notevolmente la comprensione delle relazioni e del significato delle parole all'interno di una frase.

2.1.3 Metriche di similarità

Introdotte le differenti tipologie di rappresentazioni vettoriali con il quale possiamo trovarci ad operare è bene analizzare i metodi per confrontare vettori diversi ed ottenere un indice di similarità tra di essi. Nei task di NLP infatti assume un ruolo curiale il meccanismo di confronto dell'informazione cablata all'interno dei vettori, intuitivamente l'obiettivo è dato un vettore detto di query e un assieme di vettori che rappresentano i nostri documenti restituire il sub set di elementi che sono più simili alla query data.

Vi sono differenti approcci [6] per fare questo e sono tutti basati su intuizioni geometrico spaziali che permettono di calcolare la distanza in uno spazio multi dimensionale (le dimensioni sono definite dal numero di features dei vettori che entrano in gioco) ottenendo una metrica di similarità.

Cosine similarity

La misura più comune è la cosine similarty, come suggerisce il nome permette di calcolare la similarità tra due vettori come la dimensione del coseno dell'angolo prodotto tra i due, tanto più sono simili tanto più il coseno dell'angolo tenderà ad 1.

La cosine similarity tra due vettori w e v si può calcolare come:

$$cosine(v, w) = \frac{vW}{|v||w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{2.5}$$

Per alcune applicazioni, normalizziamo preventivamente ciascun vettore dividendo per la sua lunghezza, creando così un vettore unitario di lunghezza 1. Quindi potremmo calcolare un vettore unitario da "a" dividendo per |a|. Per i vettori unitari, il prodotto scalare è uguale al coseno.

Il valore del coseno varia da 1 per i vettori che puntano nella stessa direzione, attraverso 0 per i vettori ortogonali, fino a -1 per i vettori che puntano in direzioni opposte. Ma poiché i valori di frequenza grezzi sono non negativi, il coseno per questi vettori varia da 0 a 1.

Vediamo come il coseno calcola quale delle parole "cherry" o "digital" è più vicina in significato a "information", utilizzando solo i conteggi dalla tabella abbreviata seguente: Andando a calcolare la cosine similarity tra i vettori di "information" e

	computer	data	pie
cherry	2	8	442
digital	1670	1683	5
information	3325	3982	5

Tabella 2.3: Vettori di co-occorrenza per tre parole nel corpus di Wikipedia, mostrando 3 delle dimensioni

di "cherry" ne otteniamo quanto segue:

$$cosine(cherry, information) = 0.18 \tag{2.6}$$

Calcolando invece la cosine similarity tra i vettori di "information" e di "digital" otteniamo:

$$cosine(digital, information) = 0.996 \tag{2.7}$$

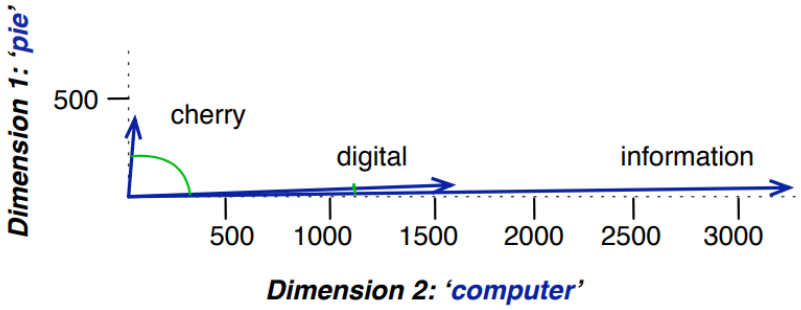

Figura 2.3: Una dimostrazione grafica della cosnine similarity, mostrando i vettori per tre parole (cherry, digital e information) nello spazio bidimensionale definito dai conteggi delle parole "computer" e "pie". Esempio tratto da [2]

Euclidean Distance

La distanza euclidea è anch'essa una metrica che viene utilizzata per valutare la similarità di due punti, nasce dall'intuizione della distanza di Minkowski, formula generica che viene declinata a diversi ordini.

$$Dis_p(x, y) = (\sum_{j=1}^{d} |x_j - y_j|^p)^{\frac{1}{p}} = ||x - y|| \tag{2.8}$$

Nel caso specifico della distanza Euclidea la p viene istanziata a 2, infatti è anche conosciuta come 2-norm dinstance. La formulala cambia come segue:

$$Dis_2(x, y) = \sqrt{\sum_{j=1}^{d}(x_j - y_j)^2} = \sqrt{(x-y)^T(x-y)} \tag{2.9}$$

Questa particolare formula ha una serie di proprietà utili in moltissimi dei task in cui è applicata tra cui il fatto di essere convesso nello spazio vettoriale. Esistono altre istanziazioni della distanza di Minkowski che permetto di catturare differenti aspetti.

2.2 Vector Database

La rappresentazione dell'informazione in forma vettoriale ha guidato la trattazione sino ad ora, in questa sezione introduciamo l'elemento che lega la rappresentazione vettoriale alla gestione e scalabilità dei dati. I database vettoriali nascono per questo scopo, trattare e rendere facilmente fruibile grosse quantità di dati rappresentate in forma di vettore numerico, sfruttando le metriche di similarità trattate alla sezione 2.1.3 per eseguire interrogazioni.

Per comprendere come si è arrivati a realizzare questo tipo di architettura è bene fare, anche in questo caso, qualche passo indietro, cercando di fare un parallelismo con database che sfruttano approcci classici come i relazionali e i NoSQL.

2.2.1 Come nascono e perché

Database Relazionali

Comunemente quando si immagina una base dati il primo pensiero è quello in un insieme di informazioni strutturate e organizzate in tabelle le quali sono relazionate tra loro in base alla business logic dell'applicativo. Questo approccio classico è ancora molto in voga e permette di gestire un gran numero di applicazioni, specialmente viene scelto per applicativi verticali su un ambito specifico in cui la fase di modellazione del dato è l'attore principale del processo.

I database relazionali dunque non sono altro che tabelle composte da colonne (attributi) e righe (le informazioni) identificate univocamente da una chiave che permette di relazionare tabelle differenti, il database attua l'integrità referenziale nelle relazioni tra le tabelle. Gli RDBMS tradizionali inoltre supportano le proprietà ACID di un database relazionale:

- Atomicità richiede che una transazione viene eseguita completamente o non viene eseguita affatto.
- Coerenza richiede che, una volta confermata una transazione, i dati devono essere conformi allo schema del database.
- Isolamento richiede che le transazioni simultanee siano eseguite separatamente l'una dall'altra.
- Durabilità richiede la capacità di ripristinare i dati all'ultimo stato conosciuto in seguito a un guasto del sistema o a un'interruzione dell'alimentazione imprevisti.

Essendo dati organizzati e strutturati il database atta la scalabilità tipicamente in verticale, incrementando le capacità di elaborazione dell'hardware, oppure in orizzontale aggiungendo repliche per i carichi di lavoro di sola lettura.

Database NoSQL

Con l'evoluzione tecnologica e sempre più dati e utenti che richiedono informazioni i database relazionali sono stati parzialmente accantonati per far spazio ai noSQL, sistemi più efficienti e scalabili per gestire dati semi-strutturati e non. I punti di forza principali sono:

- Flessibilità: i database NoSQL offrono generalmente schemi flessibili che consentono uno sviluppo più veloce e iterativo. Il modello di dati flessibile fa dei database NoSQL la soluzione ideale per i dati semi-strutturati e non strutturati.
- Scalabilità: i database NoSQL in genere sono progettati per il dimensionamento orizzontalmente, attuato usando cluster distribuiti di hardware, invece del dimensionamento verticalmente, che avviene aggiungendo server costosi e di grosse dimensioni. Alcuni fornitori di cloud gestiscono queste operazioni dietro le quinte offrendo un servizio completamente gestito.
- Elevate prestazioni: i database NoSQL sono ottimizzati per modelli di dati specifici e schemi di accesso che consentono prestazioni più elevate rispetto ai risultati che si ottengono cercando di raggiungere una funzionalità simile con i database relazionali.
- Altamente funzionali: i database NoSQL offrono API altamente funzionali e tipi di dati che sono dedicati a ciascuno dei rispettivi modelli di dati.

Queste principali caratteristiche hanno fatto si che le principali applicazioni innovative e di largo consumo adottasse questa tipologia di rappresentazione dell'informazioni che vedremo avere molto in comune con la struttura dei database vettoriali. Architetturalmente i NoSQL non sono più organizzati in tabelle, ma posso essere rappresentati in differenti modi, i più largamente utilizzati sono:

- Chiave-valore: i database chiave-valore sono altamente partizionabili e consentono dimensionamento orizzontale a livelli che altri tipi di database non possono raggiungere. Il modello di dati chiave-valore è particolarmente adatto per casi d'uso quali i videogiochi, le tecnologie pubblicitarie e l'IoT.
- Documento: nel codice di applicazione, i dati sono speso rappresentati come oggetto o un documento di tipo JSON, un modello dati efficiente ed intuitivo per gli sviluppatori. I database di documenti semplificano agli sviluppatori la ricerca e la memorizzazione di dati in un database, usando lo stesso formato di modello di documento che usano nel codice dell'applicazione. La natura gerarchica, semi-strutturata e flessibile dei documenti e dei database dei documenti, gli permette di evolversi in base alle esigenze delle applicazioni. Il

modello di documento funziona correttamente con cataloghi, profili utente e sistemi di gestione del contenuto in cui ogni documento è unico e si evolve nel tempo.

- Grafo: lo scopo di un database a grafo è facilitare la creazione e l'esecuzione delle applicazioni che operano con set di dati ad elevata connessione. I casi d'uso tipici di un database a grafo includono i social network, i motori di raccomandazione, il rilevamento di frodi e i grafi della conoscenza.

Questo cambio di approccio alla rappresentazione dei dati ha aperto le porte all'introduzione dei vector database, infatti possiamo intenderli come sotto-famiglia dei NoSQL essendo anch'essi caratterizzati da dati non strutturati rappresentati come documenti di tipo JSON. Ciò che li caratterizza è il paradigma del dato che contengono, infatti non si ha più una rappresentazione human-like, ma vettoriale.

In un database vettoriale, i dati vengono automaticamente organizzati spazialmente in base alla similarità dei contenuti, e questa similarità riguarda il significato dei contenuti piuttosto che solo le parole chiave. Se volessimo utilizzare un esempio per capire come effettivamente lavorano andando a rappresentare una biblioteca, tutti i libri verrebbero automaticamente organizzati nel database vettoriale in base al genere, anche se il genere non è stato definito in precedenza in maniera esplicita.

Ciò avviene perché i libri vengono modellati insieme come vettori densi capaci di comprendere il contesto. Inoltre, i database vettoriali consentono un alto grado di granularità nelle similarità, quindi i libri possono anche essere cercati in base allo stile di scrittura dell'autore, alle trame delle storie e altro ancora. Il tutto all'interno della stessa struttura di archiviazione, senza che si debba etichettare o contrassegnare manualmente il contenuto stesso per ricerche precise che non hanno un contesto di riferimento.

La vertiginosa crescita di modelli di intelligenza artificiale basati su embeddings ha reso obbligatorio un cambio di paradigma nella gestione e organizzazione dei dati rendendo di fatto i vector database una delle tecnologie più interessanti e utili in questo momento.

2.2.2 Architettura

All'interno i un database vettoriale i dati sono caratterizzati da tre elementi principali:

1. ID: un modo per identificare univocamente il singolo vettore all'interno della collezione

2. Embedding: la reale rappresentazione del dato caratterizzato da una dimensione fissata e coerente per ogni songolo vettore all'interno della stessa collezione, il numero di elementi del vettore può essere arbitraria e dipende dal modello che viene utilizzato per trasformare i dati grezzi in embeddings
3. Payload: Una struttura dati JSON che permette di memorizzare meta dati a corredo dell'embedding realtivo utile per aumentare il valore informativo

Figura 2.4: Rappresentazione schematizzata di come viene rappresentata l'informazione all'interno di un vector database

La figura 2.4 permette visualizzare graficamente quanto detto, ogni singolo vettore all'interno dell'architettura dei database vettoriali viene definito "punto". Questo perché concettualmente il database vettoriale non è altro che un insieme di punti all'interno di uno spazio N-dimensionale.

L'Architettura dei DB vettoriali è più complessa e l'insieme dei punti sono solo un piccolo elemento, la figura 2.5 permette di avere un overview di tutti gli elementi

che lo compongono e di come sono organizzati tra loro, i principali che si notano sono:

- Archiviazione: Rappresenta lo spazio fisico in cui viene effettivamente salvata l'informazione che contiene il vector database, comunemente, l'archiviazione può essere in memoria (memorizza tutti i vettori in RAM, ha la velocità più elevata poiché l'accesso al disco è richiesto solo per la persistenza) oppure di tipo Memmap (crea uno spazio di indirizzamento virtuale associato al file su disco).
- Collections: Una collezione è un insieme di punti (vettori che rappresentano il dato) tra i quali puoi effettuare ricerche. Il vettore di ciascun punto all'interno della stessa collezione deve avere la stessa dimensionalità e deve essere confrontato mediante una singola metrica di similarità.
- Similarity Metrics: Sono le metriche che vengono utilizzate per confrontare tra loro i vettori di una collezione e sono definite assieme alla definizione della collezione stessa.
- Points: Sono l'effettiva rappresentazione dei dati esposta e rappresentata in figura 2.4

Esternamente all'architettura del database si notano due elementi, i dati grezzi e il modello che si occupa dell'embedding, infatti la trasformazione da dati a vettori è demandata completamente a strumenti esterni a discrezione dello sviluppatore così da lasciar spazio a scelte tecniche specifiche in relazione al dominio che si sta trattando. Fondamentale però che il modello rimanga il medesimo per tutti i vettori della collezione e per, in un secondo momento, fare l'embedding delle query che volgiamo utilizzare per cercare elementi nel DB. Ogni database vettoriale declina l'architettura proposta in differenti modi, ciò che rimane tuttavia coerente è il concetto alla base e l'organizzazione delle componenti, si possono trovare DB che realizzano le collections come oggetti veri e propri in cui si possono inserire metodi e attributi arricchendone il potere informativo.

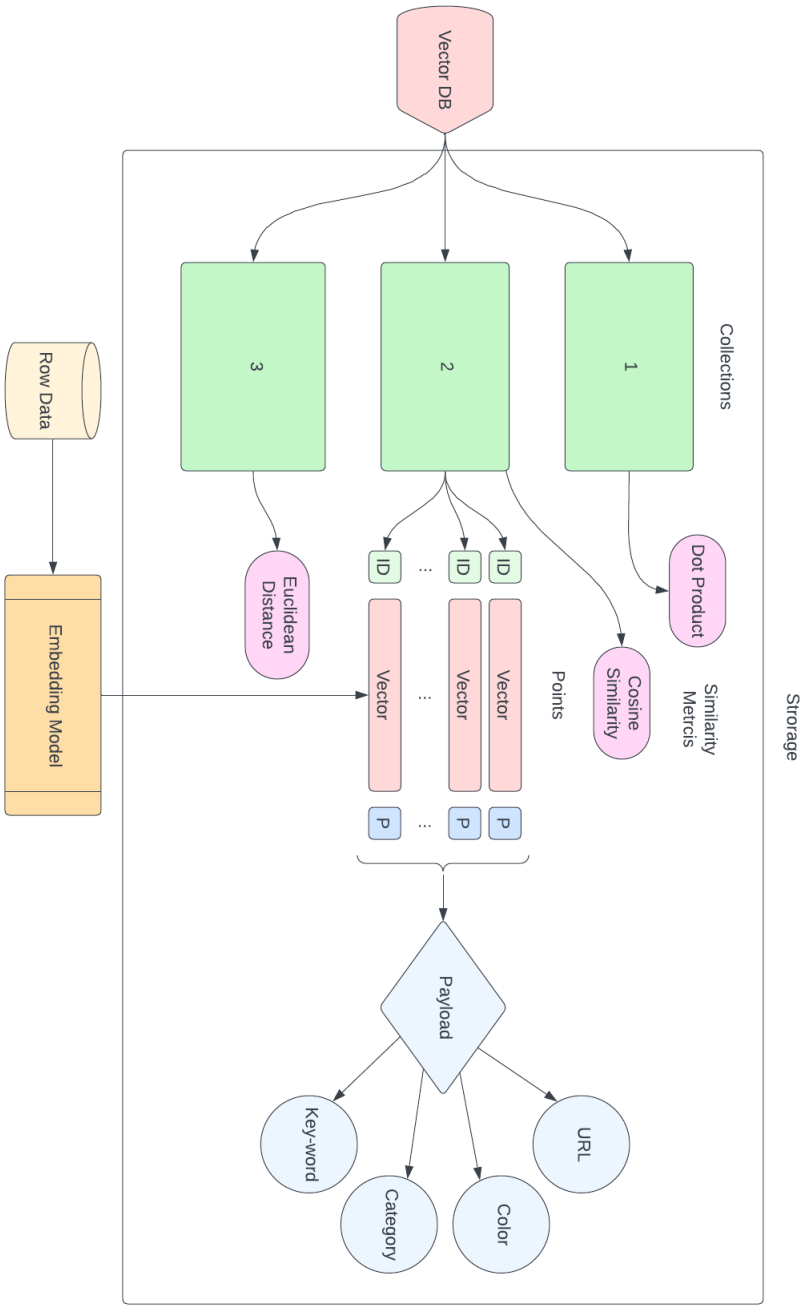

Figura 2.5: Rappresentazione dettagliata dell'architettura interna di un database vettoriale e della sua organizzazione

2.2.3 Utilizzo

I database vettoriali si calano bene in diversi use case reali in questa sezione verranno analizzati alcuni scenari in cui l'utilizzo di questa tecnologia può semplificare di molto e rivelarsi cruciale.

1. Natural Language Processing (NLP): Ad esempio, in un sistema di chatbot per il supporto clienti, ogni richiesta viene trasformata in vettore mediante l'uso di un modello di embedding. Quando un utente chiede: "Come faccio a reimpostare la mia password?", il database vettoriale può identificare query semanticamente simili come "Passaggi per il cambio della password" per fornire una risposta pertinente anche se la formulazione esatta non è presente nel sistema.

2. Sistemi di raccomandazione: Sia che si tratti di film, musica o prodotti di e-commerce, i sistemi di raccomandazione spesso si basano sulla comprensione della similarità tra le preferenze degli utenti e le caratteristiche degli articoli disponibili nel catalogo. I database vettoriali possono accelerare questo processo, rendendo le raccomandazioni personalizzate in tempo reale.

   Ad esempio, su Netflix, film e programmi TV sono rappresentati come vettori che catturano i loro generi, attori e recensioni degli utenti. Quando un utente guarda un thriller psicologico con un attore specifico, il database vettoriale può suggerire altri film dello stesso genere o film con lo stesso attore, offrendo un'esperienza di visione su misura.

3. Financial Services: In questo contesto specifico dati ad alta dimensionalità possono derivare da portafogli di investimento, modelli di trading o profili di rischio nel campo financial. I database vettoriali consentono ricerche rapide di similarità, il che è vantaggioso per la rilevazione delle frodi o per le attività di gestione del portafoglio.

   Ad esempio, i modelli di transazione degli utenti vengono rappresentati come vettori in una piattaforma bancaria. Se un utente di solito effettua acquisti di piccole dimensioni a livello locale e improvvisamente viene effettuata una grande transazione internazionale, il database vettoriale può identificare rapidamente questa operazione come anomala essendo molto diverso dal vettore proto-tipico generato su quello specifico utente, segnalandolo per una potenziale indagine sulla frode.

4. Image and Video Recognition: Dato l'aspetto ad alta dimensionalità delle immagini e dei video, i database vettoriali sono naturalmente adatti a compiti come la ricerca di similarità all'interno dei dati visivi. Ad esempio, le aziende con ampi database di immagini possono utilizzare i database vettoriali per

trovare immagini simili, facilitando compiti come il rilevamento di duplicati o la categorizzazione delle immagini.

Consideriamo una piattaforma come Pinterest[6]. Gli utenti spesso inseriscono immagini senza descrizioni dettagliate. Un database vettoriale può rappresentare ciascuna immagine come un vettore ad alta dimensionalità. Quando un utente inserisce un'immagine di un tramonto sulla costa, il sistema può cercare nel suo database vettoriale per suggerire immagini simili, forse altre vedute di spiagge o tramonti, migliorando la scoperta di contenuti e l'interazione dell'utente.

Questi citati sono solo una piccola parte di casi d'uso di questa tecnologia, ma che rendono tangibilmente l'idea del perché siano nati e in che contesti possano essere sfruttati. Nel capitolo 4 della tesi verrà affrontato un caso d'uso reale che sfruttando i vector database e che ha portato alla realizzazione di una POC.

2.2.4 Qdrant

Nel mercato ci sono molteplici proposte di Database vettoriali disponibili gratuitamente e non, in questa tesi verrà introdotto ed utilizzato Qdrant. La scelta è stata fatta per due motivi principali, il primo è che data la sua natura open source è stato possibile oltre che utilizzarlo approfondirlo tecnicamente. Il secondo punto che mi ha portato a scegliere questo prodotto è la mancanza di una ricerca ibrida nativa su cui la fase sperimentale si concentrerà.

Qdrant come architettura e gestione degli elementi si allinea a quanto presentato e descritto all'interno della sezione 2.2.2 tuttavia possiede alcune caratteristiche peculiari. Qdrant permette di realizzare collezioni multi-vettore, ogni record dunque può contenere N named vector differenti identificati univocamente dal nome e caratterizzati dalla metrica di similarità e dal numero di dimensioni. Sfruttando questa features dunque si possono avere all'interno della stessa collection più rappresentazioni vettoriali differenti che sfruttano metriche e dimensioni eterogenee. Bisogna comunque evidenziare che sebbene nemed vector differenti possono avere dimensione differenti, stessi named vector devono essere coerenti con la definizione iniziale.

Per quanto riguarda la parte di ricerca il DB vettoriale espone una specifica API per effettuare query su una specifica collezione, la ricerca può essere effettuata su più namned vector e deve sempre contenere un punto (o vettore) di dimensioni coerente con quelle della collezione per namned vector. Qdrant non farà altro che calcolare la similarità, utilizzando la metrica di distanza definita a livello di collezione, con

[6]Social network di immagini e video. pinterest.com

il/i punti valorizzati all'interno della chiamata API. Assieme all'embedding che si vuole confrontare con la base dati è importante andare a specificare anche un limite di risultati. Potenzialmente infatti il database vettoriale data una query può estrarre tutti i vettori all'interno della collezione ordinandoli da i più simili (score vicino ad 1) a quelli meno simili (score vicino a 0), dunque specificando un limite automaticamente restituisce solo gli N record con similarità maggiore alla query passata.

La ricerca tuttavia accetta altri parametri che lavorano sul payload, infatti è possibile andare a filtrare i risultati sulla base dei meta-dati presenti andando di fatto a replicare quello che in SQL classico è la clausola di where con gli operatori logici AND, OR e NOT. Qdrant permette di applicare le logiche di filtering sfruttando i seguenti operatori:

1. Must: Quando si utilizza "must", la clausola diventa vera solo se ogni condizione elencata all'interno di "must" viene soddisfatta. In questo senso, "must" è equivalente all'operatore "AND".

2. Should: Quando si utilizza "should", la clausola diventa vera se almeno una delle condizioni elencate all'interno di "should" viene soddisfatta. In questo senso, "should" è equivalente all'operatore "OR".

3. Must Not: Quando si utilizza "must_not", la clausola diventa vera se nessuna delle condizioni elencate all'interno di "must_not" viene soddisfatta. In questo senso, "must_not" è equivalente all'espressione (NON A) E (NON B) E (NON C).

Naturalmente, come su SQL, le clausole di filtering possono essere applicate assieme realizzando logiche complesse per escludere specifici elementi della nostra collezione.

2.2.5 Limiti e punti aperti

I database vettoriali come tutte le tecnologie non sono perfetti ma presentano alcune criticità:

- Complessità della struttura dei dati: La gestione di dati vettoriali complessi può richiedere un design di database più sofisticato. La definizione delle dimensioni dei vettori, la gestione delle relazioni tra vettori e l'indicizzazione efficiente possono essere sfide complesse.

- Dimensionalità elevata: I database vettoriali possono affrontare problemi di dimensionalità elevata. Con un numero elevato di dimensioni, diventa difficile visualizzare i dati e può verificarsi la "maledizione della dimensionalità", in cui la densità dei dati diminuisce drasticamente, rendendo difficile la ricerca significativa.

- Ricerca efficiente: La ricerca in database vettoriali può richiedere complessi algoritmi di ricerca, soprattutto quando si lavora con dati ad alta dimensionalità. L'indicizzazione e la ricerca efficienti possono richiedere notevoli risorse computazionali.
- Memoria e spazio di archiviazione: Dati vettoriali possono richiedere una quantità significativa di spazio di archiviazione, soprattutto se sono ad alta dimensionalità o se si stanno gestendo grandi quantità di dati. La memorizzazione e la gestione efficiente possono essere una sfida.

Una mancanza specifica del vector database Qdrant come detto è la ricerca ibrida, la successiva sezione tratterà differenti approcci nel tentativo di migliorare la classica ricerca semantica.

Capitolo 3

Hybrid Search

La ricerca ibrida è un particolare approccio all'information retrieval che permette di unire le parti migliori di due tipologie di ricerche basate sulla rappresentazione vettoriale dell'informazione. Come introdotto nella sezione 2.1 l'informazione può essere rappresentata in differenti modi, nello specifico all'interno del vector space possiamo trovarla sotto forma di vettore denso definito anche embeddigns oppure come vettore sparso che normalmente contiene le frequenze dei termini all'interno dei documenti che compongono il corpus. Spesso focalizzarci nell'utilizzare solo una delle due ricerche non è ottimale perché non permette di ottenere il risultato desiderato in termini di prestazioni, le due rappresentazioni hanno particolarità che le rendono più o meno performanti in differenti ambiti. Le aspettative sono quelle di riuscire a notare un miglioramento dei risultati dimostrando che rappresentazioni differenti colgono aspetti e particolarità diverse da dati di partenza comuni.

Questo capitolo ha come obiettivo quello di esplorare possibili approcci sia dal punto di vista teorico che dal punto di vista sperimentale per approcciare la ricerca ibrida sfruttando il database vettoriale Qdrant[1], che si basa fortemente sulla similarità tra vettori densi. Tutto questo ha l'obiettivo ultimo di migliorare l'accuratezza dei risultati prodotti per una query utente. La trattazione di questa sezione cerca di analizzare da un punto di vista critico i risultati ottenuti sperimentalmente cercando di sviscerare le motivazioni dei risultati positivi e di quelli negativi.

[1] https://qdrant.tech/

3.1 Approcci classici all'hybrid search

Il primo passo nell'ambito della valutazione e sperimentazione di diverse proposte per la ricerca ibrida è stato lo studio degli argomenti e degli approcci attualmente presenti e utilizzati nella letteratura, anche in contesti differenti da quello che verrà trattato. Non esiste un unico approccio alla ricerca ibrida, ma differenti che si calano correttamente in contesti diversi, ciò che accomuna le strategie che elencheremo a breve sono l'utilizzo contestuale dell'informazione rappresentata da vettori densi e da vettori sparsi permettendo di rendere più efficiente l'inforamtion extraction cercando di unire la parte positiva dei due.

In generale uno degli approcci maggiormente utilizzati per la gestione della ricerca ibrida in casi real time, come ad esempio la ricerca all'interno di un web browser, è una strategia a fallback. La ricerca basata su vettori densi è generalmente più costosa computazionalmente però permette di trovare associazioni semantiche, una gestione classica è lo switch automatico tra ricerca ibrida e ricerca semantica in base alla numerosità di risultati.

Quando una ricerca keyword based non produce un numero di risultati sufficiente viene arricchita con la ricerca semantica che va ad estendere il numero di risultati. La ricerca basta su vettori densi, infatti, lavorando a livello semantico con metriche di similarità raccoglie moltissimi risultati ordinati e generalmente viene scelto solo il sub-set più rilevante eseguendo una sorta di TOP N.

I motori di ricerca utilizzano anche un'interessante gestione dei vettori densi e dei vettori sparsi, partendo dalla query utente e analizzandola vengono realizzati due vettori differenti che rappresentano sia la query che i documenti del corpus da cercare:

1. Il primo vettore, denso, contiene i termini comuni, che appaiono in molti documenti;

2. il secondo invece è un vettore sparso e contiene i termini meno comuni, che compaiono solo in pochi documenti

Dobbiamo sottolineare che in questo processo i termini possono apparire sia nei vettori densi che nei vettori sparsi questo approccio specifico permette di trovare risultati più accurati, pertinenti e completi possibili.

3.2 Hybrid search per Vector Database

Spostandoci nel contesto dei database vettoriali l'idea alla base della ricerca ibrida prende spunto da quanto trattato in precedenza. L'informazione all'interno del DB vettoriale ha una rappresentazione per embeddigs dunque sfrutta vettori densi che

rappresentano la semantica di ciò che viene salvato al loro interno. I vettori sparsi e di conseguenza la ricerca per keyword richiedono la rappresentazione della medesima base documentale con un approccio basato su termn frequency e successivamente utilizzo di metriche di similarità per ottenere uno score.

Per implementare questa tipologia di ricerca all'interno dei vector database possiamo sfruttare due strategie differenti, che si basano sulla ricerca con vettori densi e rappresentazione key-word based, ciò che le rende leggermente diverse è la gestione dei risultati e della loro aggregazione.

3.2.1 Cross encoder

Nella letteratura [7] [8] la metodologia principale proposta è riassumibile con l'immagine sottostante. L'approccio hybrid search con cross encoder si basa

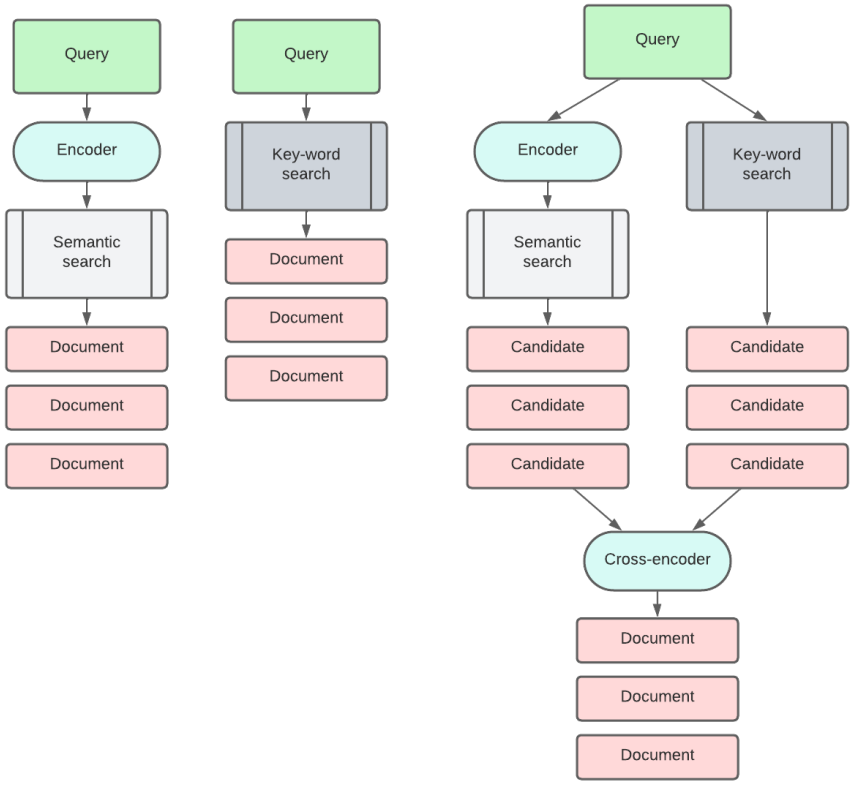

Figura 3.1: Rappresentazione grafica dell'architettura per realizzare l'Hybrid search con il cross encoder [9]

sull'esecuzione parallela di due modelli differenti di ricerca all'interno del corpus, uno basato su rappresentazione dell'informazione in vettori densi che nel caso

specifico sarà fatto sul DB Qdrant e la seconda come vettori sparsi. I due differenti metodi partendo dalla medesima interrogazione producono in modo completamente indipendente i risultati attuando strategie differenti.

Sulla base dei due gruppi di risultati candidati distinti si adopera una funzione di corss-encoding per fare una sorta fusione dei risultati, questa funzione può essere implementata in differenti modi andando di fatto a favorire una ricerca piuttosto che l'altra.

Questa strategia è una delle più utilizzate perché utilizzando due metodi completamente slegati e implementando opportunamente la funzione di cross-encoding è possibile andare a prendere il buono delle due ricerche distinte andando a migliorare l'accuratezza della ricerca. Questo è un punto cruciale e gli approcci percorsi sono trattati alla sezione 3.2.1

Capendo a fondo cosa colgono le rispettive rappresentazioni, l'idea di poter riuscire a captare dallo stesso corpus aspetti differenti che si aiutano a vicenda ed arricchiscono la collezione è l'obbiettivo desiderato. Con questo approccio i documenti rappresentati come vettori spari saranno più sensibili all'apparizione di parole specifiche come, ad esempio, acronimi, sigle e nomi specifici mentre i documenti prodotti dalla ricerca su vettori densi carpire il concetto latente dietro al documento. Sicuramente per favorire una ricerca piuttosto che l'altra servirebbe anche una certa conoscenza del dominio target. Query utente puntuali ricche di sinonimi rendono più complesso l'approccio a key-word based.

3.2.2 Classic Re-rank

Differentemente dalla precedente implementazione in questa casistica non si eseguono più in modo distinto 2 ricerche e successivamente si fondono i risultati, ma bensì si favorisce la ricerca semantica andando successivamente a ordinare i risultati sulla base delle parole chiavi presenti all'interno della query utente e dei documenti prodotti.

La scelta di favorire la ricerca semantica è dovuta dal fatto che generalmente fornisce in output molti risultati, ordinati per similarità. Essi dunque forniscono la successiva base documentale da cui partire per la seconda ricerca, quella basata a keyword.

Ciò che si andrà a fare ottenuti i risultati candidati della ricerca basata su vettori sparsi sarà eseguire un re-rank sulla base delle dei risultati differenti. Questa specifica implementazione è molto più efficiente dal punto di vista prestazionale perché la seconda ricerca è eseguita solo su un sub-set di dati e non sull'intero modello. Rispetto alla precedete proposta, in questo caso è possibile sfruttare al massimo le potenzialità del DB vettoriale andando a salvare all'interno del payload le keyword.

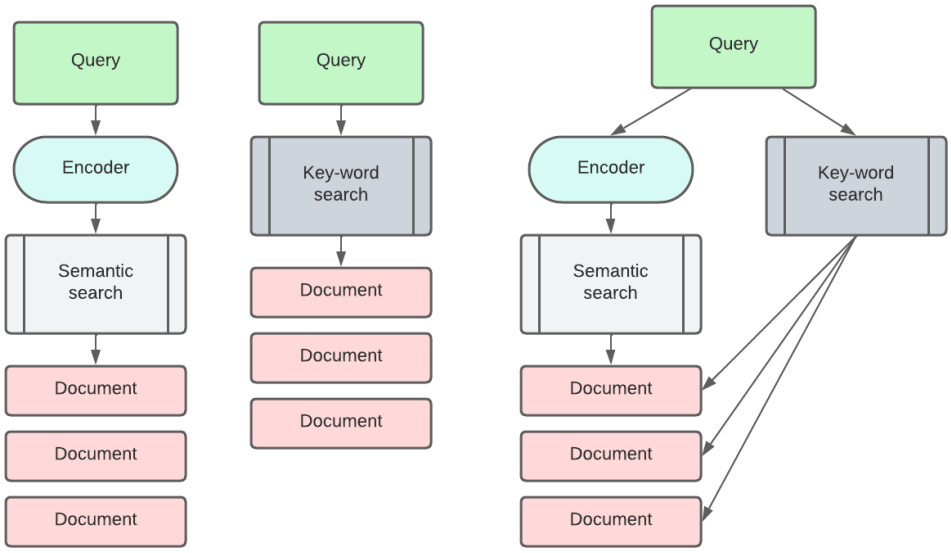

Figura 3.2: Rappresentazione grafica dell'architettura per realizzare l'Hybrid search andando a riordinare i risultati sulla base delle key-word

In questo approccio la parte critica è trovare un modo per valutare le parole chiave e di conseguenza spostare nel ranking i documenti candidati, nella sezione 3.3.7 verranno analizzati i risultati sperimentali di questa implementazione e verranno affrontate differenti approcci per cercare di trovare ma miglior funzione di re-rank.

3.3 Hybrid search con Qdrant

Il database vettoriale Qdrat già approfondito nel dettaglio alla sezione 2.2.2 non possiede nativamente l'Hybrid search, il lavoro svolto che verrà trattato all'interno di questa sezione punta a sperimentare differenti metodi per implementarla e valutarne le prestazioni analizzando e confrontando i risultati.

L'approccio scelto per cominciare la trattazione è di tipo cross encoder spiegato precedentemente nella sezione 3.2.1, per quanto riguarda la parte di ricerca basata su vettori densi è stato utilizzato Qdrant. Per ogni sperimentazione è stata realizzata una collection all'interno del Database caratterizzata da una dimensione dei vettori e una metrica di similarità. Il corpus è stato interamente trasformato in embeddigns sfruttando il modello open source all-MiniLM-L6-v2[2].

[2]https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

Sentence Transformer

Il modello scelto per gli embeddings è un Sentence Transformer[3], mappa frasi e paragrafi in vettori densi di 384 dimensioni nello spazio vettoriale ed è stato realizzato per task come il clusterng o la semantic search.

Il modello è stato addestrato in modalità self-supervised contrastive learning objective e questo ha permesso di catturare informazioni utili dai dati senza la necessità di etichette di classe supervisionate. Questo ha reso l'apprendimento più efficiente e consente di utilizzare grandi quantità di dati non etichettati per migliorare le prestazioni del modello.

Il training set del modello è stato di oltre un miliardo di frasi utilizzando dataset differenti, i principali utilizzati sono stati: Reddit comments (2015-2018)[4], S2ORC Citation pairs (Abstracts)[5], WikiAnswers Duplicate question pairs[6].

La ricerca basata su vettori, che può essere nativamente implementata con differenti metriche, produce N documenti candidati e rappresenta sostanzialmente un esecuzione classica della ricerca base implementata all'interno del vector database Qdrant, la trattazione segue con una serie di approcci alla keyword search volta a produrre altri documenti candidati da combinare per riuscire a migliorare la baseline della ricerca semantica di default proposta dal sistema.

3.3.1 Approccio TF-IDF

Come prima sperimentazione della ricerca keyword-based è stata utilizzata come funzione di peso la TF-IDF [10] acronimo di Term Frequency - Inverse Document Frequency. Questa è una delle formule più famose in letteratura ed anche una delle più utilizzate, da un'indagine condotta nel 2015 è risultato che l'83% dei sistemi di raccomandazione basati sul testo nelle biblioteche digitali utilizza tf-idf[7].

La TF-IDF permette di misurare l'importanza di un termine dato rispetto ad un insieme di documenti o ad un singolo documento, permettendo dunque di andare ad attuare una strategia di keyword search. La funzione aumenta proporzionalmente a quante volte lo specifico termine appare nel documento, ma cresce in modo inversamente proporzionale con la frequenza del termine all'interno della collezione. Questo comportamento intuitivamente andrà a favorire tutti quei termini che si

[3]Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

[4]https://github.com/PolyAI-LDN/conversational-datasets/tree/master/reddit

[5]https://github.com/allenai/s2orc

[6]https://github.com/afader/oqa#wikianswers-corpus

[7]Research-paper recommender systems: a literature survey

appaiono spesso all'interno di un documento, ma che compaiono poco nell'insieme totale dei documenti. Se si ignorasse la seconda parte termini poco significativi come le stop-word verrebbero primate perché appaiono molto di sovente tuttavia non contengono valore informativo. Ciò che una buona funzione deve fare è dare più peso ai termini che contengono significato attuando strategie come la IDF (3.2) per screditare parole di contorno. Matematicamente la funzione TF-IDF, come suggerisce il nome, si può scomporre in due fattori:

1. Il primo fattore della funzione rappresenta il numero di termini presenti nel documento. Questo valore è buona norma andarlo a normalizzare per la lunghezza totale del documento rendere confrontabili documenti di lunghezza differente confrontabili.
$$TF_{i,j} = \frac{n_{i,j}}{|d_j|} \tag{3.1}$$
dove:

$n_{i,j}$ = denota il numero di occorrenze per il termine i nel documento j
$|d_j|$ = rappresenta la dimensione, espressa in numero di termini del documento j

2. Il secondo fattore della funzione invece rappresenta l'importanza generale del termine i all'interno dell'intera collezione.
$$IDF_i = log_{10}\frac{|D|}{|\{d : i \in d\}|} \tag{3.2}$$
dove:

$|D|$ = denota il numero di documenti nell'intera collezione
$|\{d : i \in d\}|$ = rappresenta il numero di documenti che contengono il termine i

La TF-IDF ha differenti implementazioni classiche, quella utilizzata in questo caso specifico è la versione proposta da scikit-learn[8], essa differisce leggermente dalla classica formulazione andando ad inserire un termine di normalizzazione per evitare la divisione per 0.

Gli algoritmi di apprendimento automatico spesso utilizzano dati numerici, quindi quando si tratta di dati testuali o di qualsiasi attività di elaborazione del linguaggio naturale (NLP), un sotto-campo dell'IA/ML che si occupa del testo, quei dati devono prima essere convertiti in un vettore di dati numerici attraverso un processo noto come vettorizzazione. Il vettore TF-IDF contiene il calcolo del

[8]scikit-learn.org/

TF-IDF score per ogni singolo termine presente nel corpus, ogni specifico documento conterrà i punteggi dei termini che contiene. Pertanto, ciascun documento presente nel corpus avrà uno specifico TF-IDF vector, e il suddetto ha un TF-IDF score per ogni singola parola presente nell'intera raccolta di documenti. Una volta ottenuti questi vettori, è possibile applicarli a vari casi d'uso, come ad esempio verificare se due documenti sono simili confrontando il loro vettore TF-IDF utilizzando la cosine similarity.

Quello che è stato fatto nella fase sperimentale è proprio andare a rappresentare il dataset come vettorizzazione TF-IDF e successivamente andare a calcolare la similarità mediante cosine similarty con la medesima rappresentazione della query a cui vogliamo associare uno specifico documento

L'aspettativa è quella che sfruttando una rappresentazione parallela dell'informazione sia della query utente che dei documenti a disposizione la similarità prodotta permetta di catturare documenti pertinenti e differenti da quelli ottenuti dalla ricerca che sfrutta il database vettoriale Qdrant. Nella sezione 3.3.7 verranno trattati i risultati del singolo utilizzo della TF-IDF per avere un punto di partenza, mentre nella sezione 3.3.7 approfondiremo la ricerca ibrida sfruttando la ricerca del database Qdrant.

3.3.2 Approccio BM25

La successiva sperimentazione di ricerca basata su rappresentazione vettoriale sparsa è stata fatta utilizzando l'algoritmo di ranking BM25 (Best Match 25) che rappresenta in questo momento lo stato dell'arte per quanto riguarda l'information retrieval. BM25 [11] è un modello di classificazione term-based che mira a fornire risultati di ricerca accurati e pertinenti annotando i documenti in base alle loro frequenze di termine e alla loro lunghezza. Questa definizione sembra simile a quanto già trattato in precedenza con la TF-IDF, tuttavia la BM25 ne rappresenta un'evoluzione naturale. Gli elementi che la compongono sono:

- Il primo elemento è proprio la Term Frequency che è stata ampiamente approfondita nella sezione precedente
- Un altro elemento che compone l'algoritmo BM25, che lo accomuna ancora una volta alla TF-IDF è proprio la Inverse Document Frequency, anch'essa trattata precedentemente.
- il primo elemento che rappresenta una novità è la Document Length Normalization: BM25 infatti incorpora la normalizzazione della lunghezza del documento per riuscire a mitigare l'impatto di tale lunghezza sul punteggio di pertinenza. Documenti più infatti lunghi tendono ad avere più occorrenze di un termine, portando il sistema a potenziali distorsioni nel risultato finale. La

normalizzazione della lunghezza del documento contrasta questa distorsione dividendo la frequenza dei termini per la lunghezza del documento e applicando un fattore di normalizzazione.

- In ultimo un altro elemento innovativo rispetto alla TF-IDF è Query Term Saturation: BM25 include anche una funzione di saturazione dei termini per mitigare l'impatto della frequenze di termine eccessivamente elevati all'interno di un documento. Questa funzione quindi riduce l'effetto della frequenze estremamente elevate di uno specifico termine sul punteggio di pertinenza, poiché le frequenze molto alte spesso corrispondono a termini meno informativi. Questo è in contrasto con alcune altre metriche di pesatura dei termini, come ad esempio la TF-IDF, infatti la frequenza del termine in un documento è semplicemente moltiplicata per l'inverso della frequenza del termine nell'intera collezione di documenti. La scelta di modellare la saturazione dei termini nella query è una caratteristica fondamentale di BM25 poiché contribuisce a evitare l'accentuazione eccessiva di documenti che contengono ripetizioni di un termine specifico. BM25 cerca invece di concentrarsi su documenti che contengono una varietà di termini rilevanti.

Matematicamente BM25 puo essere espressa con la seguente formula:

$$BM25(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_i,D) + k_1(1 - b + b\frac{|D|}{avgdl})} \tag{3.3}$$

dove:

D = rappresenta uno specifico documento
Q = rappresenta la query su cui volgiamo calcolare il BM25 score
$avgdl$ = identifica la Document Length Normalization
k, b = sono iper-parametri che permettono di controllare l'impatto della Document Length Normalization e Query Term Saturation sul risultato complessivo

Sfruttando questa rappresentazione vettoriale della base documentale le aspettative sono di un miglioramento tangibile rispetto alla versione che sfrutta TF-IDF essendo come spiegato una evoluzione che punta ad affinare la tecnica di scoring. Come fatto per la TF-IDF tratteremo i risultati prima del modello singolo per valutare le sue prestazioni, sezione 3.3.7. Successivamente si analizzeranno i risultati della ricerca ibrida, sezione 3.3.7.

3.3.3 Cross encoding

Una sezione cruciale della sperimentazione è stata riuscire a trovare metodo efficace per andare ad unire i risultati delle due ricerca eseguite parallelamente. Il primo

ostacolo è in termini prettamente numerici, infatti gli score assegnati dalla cosine similarity sono compresi tra 0 e 1 mentre sfruttando i modelli basati su vettori sparsi, come BM25, i punteggi di similarità sono espressi con numeri che possono essere anche maggiori di 1. Il primo approccio dunque è stato piuttosto semplicistico andando ad eseguire una normalizzazione portando il dominio di entrambe le ricerche in un intervallo tra 0 e 1. Una volta fatto questo è sufficiente ordinare il set di risultati delle due ricerche evitando naturalmente i duplicati.

Questo meccanismo non è ottimale in quanto ignora i duplicati (non andando quindi a pesarli maggiormente), inoltre non considera minimamente la posizione di apparizione nel risultato. Il primo approccio non ha portato i risultati sperati andando, in alcuni casi a degradare addirittura le prestazioni del modello. Ci si è concentrati dunque sui difetti individuati in questa prima proposta che sono la causa di una diminuzione delle prestazioni cercando di trovare una funzione di aggregazione dei risultati che tenesse conto di tutti gli elementi utili.

Approfondendo il tema andando a cercare in letteratura paper su argomenti simili, dopo svariati buchi nell'acqua mi sono imbattuto nel lavoro[9] di Benham and Culpepper [12], dove viene proposto, assieme ad altre funzioni, l'approccio RRF (Reciprocal Rank Fusion). La funzione proposta e trattata all'interno dell'articolo accademico è così strutturata:

$$\sum_{d \in D} \frac{1}{k + r(d)} \tag{3.4}$$

Questa funzione permette di andare a pesare differentemente i documenti in base alla posizione di apparizione nei rispettivi due risultati, esempio in tabella: Il risultato

Posizione	Ricerca 1	Ricerca 2	RRF
1	Doc2	Doc1	Doc2: 1/1 + 1/3 = 1.3
2	Doc1	Doc3	Doc1: 1/2 + 1/1 = 1.5
3	Doc4	Doc2	Doc4: 1/3 + 1/4 = 0.53
4	Doc3	Doc4	Doc3: 1/4 + 1/2 = 0.75

Tabella 3.1: Tabella esemplificativa del funzionamento del della funzione di fusione Reciprocal Rank Fusion per eseguire il cross-encoding delle ricerche basate su vettori densi e sparsi

quindi è una nuovo set di documenti candidati ordinati in modo coerente con l'apparizione nelle rispettive ricerche tenendo conto della loro posizione. Partendo dalla versione proposta è possibile aggiungere un iper-paramentro θ per favorire

[9] Risk-Reward Trade-off in Rank Fusion

una delle due ricerca moltiplicando, a seconda della scelta, uno dei due membri della somma. Tutta la trattazione sfrutta un θ uguale ad 1, quindi sostanzialemnte si pesano nello stesso modo i risultati.

3.3.4 Keyword re-rank

Come spiegato nella sezione 3.2.2 l'approccio del cross encoding non è l'unico, esiste infatti una metodologia per sfruttare in modo differente l'approccio keyword-based. Qdrant mette a disposizione per ogni vettore un payload in cui si possono inserire meta-dati specifici. In questa sezione tratteremo una possibile implementazione di keyword extraction partendo dai dati del corpus andando non più a salvare il semplice embeddig per ogni elemento del dataset, ma anche una serie di parole chiavi che la caratterizzano. Questo specifico approccio a differenza dei metodi precedenti permette di lavorare nativamente sul vector database non dovendo mantenere matrici sparse che rappresentano il dataset.

Come mostrato in figura 3.2 verrà eseguita in primis la ricerca basta su vettori densi che tipicamente produce molti risultati, andandola sostanzialmente a favorire rispetto alla ricerca per parole chiave. Sulla base dei risultati fin qui ottenuti si procede andando a calcolare la similarità tra le parole chiave all'interno della query di ricerca e le keyword precedentemente calcolate, questo per ogni vettore dell'output della ricerca semantica. Questo secondo passaggio permette di andare a modificare la classifica dei risultati spostando di posizioni i risultati proposti e andando a favorire risultati che possiedono keyword prenseti nella query.

KeyBERT

Come modello di keyword extraction è stato utilizzato KeyBert[10] questa libreria sfrutta l'embedding BERT, dettagliato in sezione 2.1.2. Come prima cosa, vengono estratti gli embeddings del documento con BERT per ottenere una rappresentazione a livello di documento. Successivamente, vengono estratti gli embedding delle parole per parole/frasi N-gram (la libreria permette di scegliere gli N-grammi). Infine, utilizziamo la cosine similarity per individuare le parole/frasi che sono più simili al documento. Le parole più simili potrebbero quindi essere identificate come le parole che descrivono meglio l'intero documento.

Utilizzando un approccio di questo tipo l'estrazione delle keyword risulta più robusta a termini come stop-word e segni di punteggiatura che con metodi più tradizionali, come quelli basati sulla frequenza, se non appositamente gestiti valuterebbero erroneamente come parole chiave.

[10]https://github.com/MaartenGr/KeyBERT

Questo all'atto pratico sembra una strategia interessante per riuscire a sfruttare appieno le potenzialità del database vettoriale andando non più ad avere una vera e propria doppia rappresentazione dei documenti da cui cerchiamo le informazioni, ma mantenendo a livello di payload le informazioni necessarie per eseguire il successivo re-rank sulla base delle parole chiavi. Nonostante inizialmente sembrasse una buona idea approcciare la ricerca ibrida in questo senso nella sezione di approfondimento dei risultati 3.3.7 verranno analizzate le motivazioni del perché sperimentalmente non si è rilevata una intuizione corretta.

3.3.5 Dataset Utilizzati

Il dataset scelto per andare a sperimentare le differenti ricerche e le relative implementazioni di Hybrid search è Home depot[11].

Questo dataset contiene le descrizioni di diversi prodotti e una serie di query utenti di reali termini di ricerca fatte dal sito web di Home Depot. Il dataset è etichettato e per ogni query di ricerca vi è l'etichetta del prodotto che ci aspettiamo venga trovato, Home Depot ha coinvolto molteplici valutatori umani per annotare le coppie query/prodotto.

Oltre al semplice prodotto per ogni query è presente anche la rilevanza ed è rappresentata con un numero compreso tra 1 (non rilevante) e 3 (molto rilevante). Ad esempio, una ricerca per "pila AA" sarebbe da considerarsi molto rilevante per un pacchetto di pile di dimensioni AA (rilevanza = 3), moderatamente rilevante per una pila per trapano senza cavo (rilevanza = 2), e non rilevante per una pala da neve (rilevanza = 1).

La rilevanza delle ricerca per questa trattazione è stata utilizzata per ordinare per rilevanza i risultati e riuscire ad assegnare punteggi maggiori per risultati trovati più pertinenti.

Il Dataset è presente su Keggle ed è stato presentato per una challenge in un ambito differente rispetto alla semplice ricerca di informazioni tuttavia grazie alla struttura, alla presenza di query annotate e numerosità dei dati questo dataste si è prestato bene per trattare la sperimentazione sulla ricerca ibrida.

3.3.6 Metriche di confronto

Per confrontare le performance delle varie implementazioni di hybrid search tra loro e rispetto alle semplici ricerche per keyword o semantiche sono state utilizzante differenti metriche note in letteratura per cogliere i vari aspetti delle diverse implementazioni.

[11] https://www.kaggle.com/c/home-depot-product-search-relevance

Avendo sfruttato dataset etichettati in cui per ogni query sapevamo quali fossero i risultati attesi è stato possibile andare ad utilizzare metriche come la precision e la recall.

Precision

La precision permette di valutare in un range tra 0 e 1 quanto il sistema è bravo a classificare come positivi esempi che lo sono effettivamente. Numericamente rappresenta la frazione dei veri positivi (TP che possiamo ottenere grazie alle query etichettate) rispetto a tutti gli elementi classificati come positivi dal modello (veri positivi più falsi positivi, FP). La precisione sarà tanto più vicina ad uno quanto il modello riesce ad evitare di etichettare erroneamente elementi negativi come positivi.

$$\frac{TP}{(TP + FP)} \tag{3.5}$$

Recall

La recall permette di valutare in un range tra 0 e 1 quanti dei dati realmente positivi che sono stati identificati correttamente dal modello. Numericamente rappresenta la frazione dei veri positivi (TP, che possiamo ottenere grazie alle query etichettate) rispetto a tutti gli elementi effettivamente positivi del dataset etichettato (veri positivi più falsi negativi, FN). La recall sarà tanto più vicina ad uno quanto il modello riesce ad evitare perdere troppi casi positivi reali.

$$\frac{TP}{(TP + FN)} \tag{3.6}$$

F1 Score

L'F1-Score è una misura che combina precision e recall. Essa è generalmente rappresentata come media armonica dei due, e non è solo un altro che un modo per calcolare una media di valori, generalmente descritta come più adatta per rapporti come nel caso della precision e recall rispetto, ad esempio, alla media aritmetica tradizionale. La formula utilizzata per il punteggio F1 in questo caso è:

$$2 * \frac{Precision * recall}{Precision + recall} \tag{3.7}$$

L'idea è di fornire una singola metrica che pesa i due rapporti in modo equilibrato, richiedere ad entrambi di avere un valore più elevato per aumentare il valore del punteggio F1. Ad esempio, una precisione di 0,01 e richiamo di 1,0 darebbe:

1. una media aritmetica di $\frac{0{,}01+1{,}0}{2} = 0{,}505$,

2. Punteggio del punteggio F1 di $2 * \frac{0,01*1,0}{0,01+1,0} = 0,02$.

Questo perché l'F1 score è molto più sensibile a uno dei due ingressi con un valore basso (0,01 qui). Il che rende un'ottima metrica per valutare le prestazioni generali di un modello.

3.3.7 Risultati ottenuti

In questa sezione si andranno a trattare i vari risultati ottenuti. La sperimentazione si è svolta per fasi andando ad approcciare il problema e analizzando le differenti implementazioni separatamente, producendo risultati e successivamente lavorando sull'aggregazione di essi implementando la ricerca Ibrida.

Tutti le varie esecuzioni sono valutate con le metriche scelte e trattate alla sezione 3.3.6. Si è deciso di declinare tutte e tre le metriche in due contesti differenti: le valutazioni sono state fatte sia sui primi @15 risultati che sui primi @20.

Semantic Search

Dopo aver inizializzato e configurato Qdrat tramite apposita immagine Docker è stata creata una nuova collection all'interno del database, caratterizzata da vettori con 384 punti, gli stessi prodotti dal Sentence Transformer scelto. Per quanto riguarda la metrica di similarità è stata scelta la Cosine Similarity approfondita alla sezione (2.1.3).

Sfruttando il dataset Home Depot e il Sentence Transformer all-MiniLM-L6-v2 sono stati calcolati tutti gli embeddings delle descrizioni presenti nel dataset, circa 124K, e come payload del vettore è stato inserito l'uuid del prodotto associato. Questa prima fase rappresenta la materializzazione del dataset all'interno del DB Qdrant.

Sempre fornito da Home Depot vi sono le query etichettate, che sono state utilizzate per produrre le statistiche per ogni singola implementazione, dopo aver caricato le query e le etichette all'interno di un test-set è stata eseguita la classica ricerca fornita nativamente su Qdrant, basata su cosine similarity, andando a calcolare per ogni query k documenti candidati e sulla base di queste rispetto alle etichette fornite sono state calcolare le metriche discusse nella sezione precedente.

I risultati ottenuti sono riassunti nella tabella 3.2

	Sematic Search
Recall@15	0.1983
Recall@20	0.2282
Precision@15	0.1062
Precision@20	0.0921
F1_Score@15	0.1383
F1_Score@20	0.131

Tabella 3.2: Tabella riassuntiva dei risultati raggiunti sul dataset Home Depot eseguendo la ricerca semantica con cosine similarity di Qdrant

Questi risultati saranno trattati come baseline, l'obiettivo dunque è quello di migliorare quanto propone di default il vector database.

TF-IDF

Trattiamo ora i primi risultati della ricerca basata su rappresentazione sparsa dell'informazione, TF-IDF rappresenta il punto di partenza per quanto riguarda questo approccio, nella tabella 3.3 si nota come i risultati sono più bassi rispettivamente all'esecuzione effettuata con Qdrant 3.2. Vedremo nella trattazione della ricerca ibrida basata su TF-IDF se, nonostante il risultato si riuscirà a migliorare le performance del vector database.

	TF-IDF
Recall@15	0.1063
Recall@20	0.1691
Precision@15	0.0533
Precision@20	0.0585
F1_Score@15	0.0709
F1_Score@20	0.0892

Tabella 3.3: Tabella riassuntiva dei risultati raggiunti sul dataset Home Depot applicando come metrica di similarità la cosine similarity sulla rappresentazione sparsa sfruttando TF-IDF

Hybrid search TF-IDF

In analisi i risultati ottenuti andando ad implementare la ricerca ibrida unendo i risultati prodotti dalla rappresentazione con TF-IDF e la semantic search di Qdrant, la tabella 3.4 si possono notare i risultati sperimentali ottenuti.

	TF-IDF	Semantic Search	Hybrid Search
Recall@15	0.1063	0.1983	0.1869
Recall@20	0.1691	0.2282	0.2192
Precision@15	0.0533	0.1062	0.0958
Precision@20	0.0585	0.0921	0.0892
F1_Score@15	0.0709	0.1383	0.1418
F1_Score@20	0.0892	0.131	0.1382

Tabella 3.4: Tabella riassuntiva dei risultati ottenuti andando ad eseguire la ricerca ibrida unendo i risultati prodotti da Qdrant e dal modello TF-IDF

I risultati della ricerca ibrida mostrano un leggero decremento delle prestazioni, questo è dato dal fatto che la ricerca basa su TF-IDF non riesce ad arricchire il set di dati proposto da Qdrant, anzi in alcuni applicando il cross encondig trattato in sezione 3.2.1 esclude risultati corretti dal set finale di risposte. Di per se il risultato dimostra come il modello basato su frequenze se non propriamente performante può andare a degradare le prestazioni generali del sistema, questo risultato negativo mi ha portato a valutare l'esplorazione di alternative come ad esempio BM25.

BM25

In questa sezione trattiamo i risultati ottenuti sul dataset Home depot andando ad utilizzare la ricerca Keyword based sfruttando il modello BM25.

	BM25
Recall@15	0.1552
Recall@20	0.1845
Precision@15	0.076
Precision@20	0.071
F1_Score@15	0.1020
F1_Score@20	0.1025

Tabella 3.5: Tabella riassuntiva dei risultati raggiunti sul dataset Home Depot applicando una rappresentazione sparsa sfruttando il modello BM25 e il suo scoring

Come si nota dalla tabella riepilogativa 3.5 anche in questo caso i risultati sono peggiori comparati a quanto ottenuto per la semplice ricerca semantica 3.2, ma decisamente migliorativi rispetto alla TF-IDF 3.3. I motivi di questa differenza di prestazioni rispetto alla TF-IDF sono attesi, BM25 come visto ne rappresenta la naturale evoluzione dunque è corretto aspettarsi un miglioramento tangibile anche da punto di vista sperimentale. Per quanto riguarda le differenze con la ricerca semantica essendo rappresentazioni fondamentalmente diverse subentra anche una propensione dei dati e delle query ad una piuttosto che all'altra. Il dataset scelto presenta descrizioni abbastanza ricche, ma query utente con poche parole, dove spesso sono rappresentano sinonimi dei termini che appaiono nella descrizione target. Questo giustifica una maggiore difficoltà dell'apprccio basato sulle frequenze come il modello BM25 rispetto ad una ricerca basata su embeddings. L'intera sperimentazione è stata fatta utilizzando la libreria Python BM25Okapi.

Hybrid search BM25

In questa sezione vengono presentati i risultati aggregati per le metriche valutative scelte nella sperimentazione dell'hybrid search appoggiandosi al vector database

Qdrant semantic search e BM25 come modello di rappresentazione sparso. I risultati sono stati aggregati andando ad utilizzare l'approccio trattato nella sezione 3.2.1.

	BM25	Semantic Search	Hybrid Search
Recall@15	0.1552	0.1983	0.2039
Recall@20	0.1845	0.2282	0.2396
Precision@15	0.076	0.1062	0.1088
Precision@20	0.071	0.0921	0.0972
F1_Score@15	0.1020	0.1383	0.1418
F1_Score@20	0.1025	0.131	0.1382

Tabella 3.6: Tabella riassuntiva dei risultati ottenuti andando ad eseguire la ricerca ibrida unendo i risultati prodotti da Qdrant e dal modello BM25

Dai risultati si nota che vi è un miglioramento netto rispetto alla ricerca sparsa eseguita con BM25 mentre rapportando i risultati con la classica ricerca semantica abbiamo un miglioramento del 5% circa. Questo risultato dimostra la tesi iniziale che, rappresentazioni differenti colgono aspetti differenti dai dati. Andando ad analizzare nel dettaglio porzioni di risultati si nota che la maggior parte dei risultati individuati erano comuni ad entrambi modelli, tuttavia la ricerca per keyword tendeva a possedere risultati in posizione elevate non presenti invece nel top N della ricerca semantica. Grazie all'implementazione della RRF[12] che permette di pesare correttamente i risultati eseguite distintamente otteniamo come output finale un set di documenti più ricco di risultati corretti aumentando le prestazioni generali.

Questo risultato conclude la fase sperimentale per quanto riguarda l'approccio cross-encoder della ricerca ibrida, infatti partendo dalla baseline fornitaci dal vector database Qdrant si è lavorato con differenti approcci alla rappresentazione sparsa sino ad ottenere un effettivo miglioramento delle performance generali del modello, toccando con mano quello che si era intuito studiando le rappresentazioni da un punto di vista teorico.

Hybrid search con Keyword re-rank

I risultati ottenuti andando ad utilizzare l'approccio di riclassificazione basato su parole chiavi 3.2 sono riassunti dalla tabella seguente che aggrega i risultati della ricerca basata solamente su vettori densi (colonna Semantic Search) e risultati riclassificati (colonna Hybrid Search).

Per riuscire a raggiungere un risultato accettabile è stata fatta molta sperimentazione per quanto riguarda la funzione di riclassificazione, sono state percorse

[12]Reciprocal Rank Fusion

tre strade differenti di complessità via via incrementale che hanno permesso di migliorare le prestazioni, ma non di raggiungere il risultato proposto con l'hybrid search eseguita con BM25:

1. Overlap Lessicale: l'idea di base era semplicemente di eseguire un overlap lessicale sulla base delle parole chiave trovate rispettivamente nelle query utente e nelle descrizioni, questo tuttavia non portava ad un effettivo incremento prestazionale, anzi la ricerca degradava rispetto alla versione puramente semantica. Questo è dato dal fatto che si tagliavano risultati senza andare effettivamente a pesare lo score associato ad ogni singola parola chiave e l'emissione di parole chiavi multiple. Per questo motivo è stata trascurata come strada

2. Key-word centroid: avendo ipotizzato che uno dei problemi principali nella prima versione della funzione di fusione fosse l'emissione di sinonimi come parole chiave tra i documenti e query utente si è tentato un approccio differente. Dato ogni parola chiave si calcola l'ebedding relativo, successivamente una volta ottenuti tutti i singoli emebdding delle parole chiave dei documenti e della query si calcola un centroide che ne approssima il risultato. A questo punto l'idea non è più quella di confrontare parole chiavi ma punti che rappresentano la media dei significati delle parole chiave. Fatto questo i risultati sono stati riordinati e prodotti in output. Ciò che è saltato alla luce è un sostanziale stallo delle prestazioni infatti i risultati sono in linea con quelli ottenuti dalla classica esecuzione delle ricerca semantica. La motivazione di questo è intuitiva, il calcolo del centroide eseguito sulle keyword non fa altro che dare una medesima approssimazione rispetto al calcolo dell'embedding contestuale dell'intero documento. Per quanto riguarda le query il discorso è simile, ma ancora più estremo dato che il dataset utilizzato è caratterizzato da query molto brevi in cui spesso appaiono solamente le parole chiave estratte, dunque è come se si eseguissero parallelamente due ricerche semantiche e di fatto questo non comporta miglioramenti.

3. Overlap Fusion: L'ultimo approccio tentato per andare a riordinare i risultati cerca di applicare l'intuizione precedentemente utilizzata e spiegata nel dettaglio nella sezione 3.2.1. Per ogni risultato prodotto dalla prima ricerca semantica si valuta quante sono le parole chiavi presenti rispetto alla query utente. Una volta identificato il numero e la posizione (ogni parola chiave grazie al modello KeyBERT è ordinata secondo la rilevanza) delle parole chiavi si aumenta lo score prodotto dalla prima ricerca di un fattore θ che decresce linearmente sulla base della rilevanza della parola chiave in quel documento specifico. Questo permette di andare a spostare verso l'alto tutti quegli elementi che contengono una o più parole chiave nonostante questo

modificando l'iperparametro θ è possibile di fatto far pesare di più o di meno questa riclassificazione.

	Semantic Search	Hybrid Search
Recall@15	0.1983	0.2045
Recall@20	0.2282	0.2287
Precision@15	0.1062	0.1059
Precision@20	0.0921	0.0895
F1_Score@15	0.0939	0.1082
F1_Score@20	0.0941	0.1095

Tabella 3.7: Tabella riassuntiva dei risultati dell'esecuzione sul dataset Home depot dell'Hibrid search con il vector database Qdrant con L'approccio keyword re-rank

I risultati ottenuti e riassunti nella tabella 3.7 mostrano un leggero miglioramento rispetto alla semplice ricerca semantica, ma questo approccio risulta più debole rispetto a quanto analizzato in precedenza con il cross-encoder. Avere una a disposizione solo gli N documenti prodotti dalla ricerca su vettori densi riduce di molto le possibilità di andare a ottenere, semplicemente riordinandoli sulla base delle parole chiavi, nuovi documenti che non apparivano precedentemente. Questo non dimostra una totale inaffidabilità dell'approccio proposto tenendo ancora aperta la strada a possibili miglioramenti.

Approfondendo tale approccio all'interno della letteratura, nella ricerca di risultati più incoraggianti di quanto ottenuto in fase sperimentale non sono riuscito a trovare articoli che adottano l'esatto approccio proposto nella trattazione. Infatti ragionando a posteriori, questo particolare approccio non va realmente ad effettuare una ricerca ibrida in quanto tale, ma più che altro cerca di migliorare il risultato della semplice ricerca semantica sfruttando il concetto di parole chiave rendendo più robusto il sistema.

Una possibilità per migliorare le prestazioni è quella di sfruttare Key-word centroid avanzato, invece di calcolare semplicemente la similarità tra i centroidi dei documenti prodotti dalle parole e da quello della query si potrebbe andare a confrontare il senso delle parole stesso utilizzando embedding statici, come ad esempio GloVe. Questo teoricamente dovrebbe evitare il problema riscontrato e trattato della convergenza del centroide delle keyword con il vettore che rappresenta il documento stesso aumentando di fatto di pochissimo il valore informativo su cui effettuare una riclassificazione. Utilizzando direttamente embeddings statici l'idea sarebbe simile a quella proposta con l'overlap fusion, ma basata su vettori. Andando a configurare a dovere un threshold è possibile decidere quando due

ebegging sono sufficientemente simili. Questo permetterebbe di valutare anche i sinonimi delle key-word migliorando il sistema.

A differenza della proposta basata sulla funzione per il corss-encoding Reciprocal Rank Fusion, in questo caso specifico qui riordiniamo i documenti sulla base di informazioni presenti nei documenti stessi, di conseguenza è più complesso ottenere stravolgimenti significativi nelle prestazioni. Tuttavia questo metodo potrebbe essere approfondito e sfruttato per migliorare la ricerca semantica e successivamente sfruttare il corss-encoding per migliorare ancora ulteriormente i risultati. Questa intuizione potrebbe alla fusione dei due approcci presentati e analizzati: eseguendoli di fatto in cascata l'obiettivo sarebbe quello di affinare la ricerca semantica sfruttando le keyword presenti al suo interno e successivamente utilizzare la Reciprocal Rank Fusion per combinare la base documentale ottenuta sulla rappresentazione dell'informazione come vettore sparso.

Capitolo 4

Proof of Concept

Il seguente capitolo tratta in modo dettagliato la realizzazione poroof of concept in ambito industriale sviluppato durante il mio percorso lavorativo in Clusetr Reply. L'obiettivo è quello di mostrare come i database vettoriali e più in generale la rappresentazione dell'informazione in vettori densi possano essere sfruttati per la realizzazione di proposte innovative in ambiti lontani da quelli per cui sono classicamente pensati.

All'interno della POC ho avuto la fortuna di sperimentare e utilizzate altre tecnologie tra cui due modelli di Open AI: Ada[1] per ottenere gli embeddings successivamente immagazzinati nel Database vettoriale scelto e il Large languange model GPT-4[2]. Azure è stata invece l'infrastruttura cloud utilizzata e su cui è stato eseguito il deploy dell'intera architettura del proof of concept.

Il lavoro svolto verrà introdotto trattando il contesto in cui è stato sviluppato descrivendo l'azienda e il cliente per cui è stato sviluppato e in secondo luogo analizzando in dettaglio la progettualità da cui è nata l'esigenza che la POC ha l'obbiettivo di risolvere.

[1] model: text-embedding-ada-002

[2] Generative Pre-trained Transformer 4

4.1 Contesto di sviluppo

In questa breve sezione verrà trattato il contesto aziendale nel quale è nata l'idea divenuta successivamente proof of concepet e l'ambito in cui opera il cliente a cui è stata presentata per riuscire a contestualizzare alcune scelte tecniche fatte.

4.1.1 Reply S.P.A.

Reply S.P.A. è un'azienda di consulenza informatica torinese fondata da Mario Rizzante nel 1996. Ad oggi, settembre 2023 conta circa 15mila dipendenti sparsi per tutto il mondo specialmente in Italia. Attualmente ala guida della società c'è Tatiana Rizzante, figlia del fondatore.

Reply sin dalla nascita ha avuto una struttura societaria particolare, infatti è definita come una rete di aziende dove Reply holding fa da capogruppo. Questa struttura permette alle varie company sotto Reply di specializzarsi verticalmente su due driver principali:

1. Tipologia di cliente: Automotive, Finanacial, Health...
2. Stack di sviluppo: Microsfot, Google, AWS, Oracle...

Questa particolare struttura ha permesso di crescere velocemente in ambiti relativi alla consulenza informatica molto differenti tra loro riuscendo a dare continuità di lavoro alle risorse. Reply grazie a questa organizzazione offre ai dipendenti la possibilità di muoversi internamente tra le varie company permettendo loro di vedere nuovi ambiti di sviluppo senza perdere gerarchia societaria e anni di anzianità, rendendo possibile l'acquisizione di skill orizzontali su ambiti molto differenti.

I principali ambiti in cui Reply offre consulenza sono:

- Banche, Assicurazione e Operatori Finanziari: Reply si sta concentrando molto in questo ambito focalizzandosi sulla trasformazione digitale delle istituzioni finanziarie in Europa, offrendo servizi completi per il cliente stesso o per utenti terzi come le applicazione di home banking.
- Telecomunicazioni e Media: Ambito in cui Reply è specializzata è la revisione del modello di engagement omni-canale del cliente. Il modello comprende l'insieme di tutte le varie interazioni, mediante svariati canali, per favorire la relazione con il cliente, al fine di aumentare la soddisfazione e la fedeltà del cliente finale.
- Manifacturing e Retail: Reply offre supporto alle aziende per le fasi di trasformazione e gestione dei sistemi informativi per migliorare l'esperienza cliente,

attraverso lo sviluppo di soluzioni complete. Inoltre supporta le aziende nella gestione dei proprio processi interni e gestione delle loro risorse tramite applicativi ah hoc.

4.1.2 Cluster Reply

Cluser Reply è una company del gruppo Reply specializzata principalmente in soluzioni di system integration applicati a tecnologie Microsoft sia on-premise[3] che in-cloud. L'azienda è principalmente italiana con differenti sedi in tutto i territorio, possiede inoltre una sede in Brasile e due in Germania.

L'azienda è a sua volta divisa per ambito e BU[4], nel mio caso specifico tutti i clienti appartengono all'ambito finanziario assicurativo e la mia business unit di riferimento si occupa di Data analisi e Intelligenza Artificiale. La POC trattata mostrerà come i due ambiti se pur distanti possono essere legati per risolvere una inefficienza individuata su una progettualità.

Cluster Reply possiede al suo interno altre 3 divisioni per area di pertinenza dei clienti, ma sempre legate alle tecnologie Miscrosoft nello specifico:

1. Manifacturing
2. Automotive
3. IoT & Security

4.1.3 Il cliente

Il cliente che ha richiesto lo sviluppo del progetto di business intelligence affrontato alla sezione 4.2.1 e su cui è stata proposta la POC trattata in questa sezione è una delle compagnie assicurative più grandi d'Europa e opera anche a livello finanziario con la gestione di portafogli di investimento tramite la sua controllata dedicata.

Il cliente possiede un'area tecnica interna con la quale ci interfacciamo quotidianamente per coordinare gli sviluppi e concordare requisiti e tempistiche di messa in produzione relativamente alle richieste degli utenti business. La possibilità di lavorare a stretto contatto con questa tipologia di cliente mi ha dato l'occasione di sperimentare metodologie di lavoro differenti e progettualità interessanti con un elevato grado di responsabilità e autonomia.

[3]Server e dati gestiti internamente dal cliente

[4]Businee Unit

4.2 Progetto e Obiettivo

La proposta realizzata punta ad andare a risolvere una problematica reale sollevata ed individuata assieme al cliente cercando di sfruttare al massimo le tecnologie innovative che sono presenti sul mercato. Il progetto di partenza, su di cui è nata la necessità, è decisamente lontano dalle tematiche trattate, ma è un esempio di come tecnologie innovative tipo i database vettoriali, sono duttili in molteplici contesti reali.

4.2.1 Progetto di business intelligence

Il progetto da cui è nata la POC è un progetto di business intelligence, progettualità sempre più richieste da tutti i clienti finance, questo perché permette di trarre vantaggio dai propri dati e dalla loro rappresentazione. Nello specifico il progetto ha per il cliente l'obiettivo di andare a sostituire interamente il processo interno che si occupa dello sviluppo di reportitisca e relativa Knowledge sharing tra differenti team di analisti finanziari dell'intero gruppo.

Riuscire a realizzare report e dashboard in modo semplice ed intuitivo è cruciale nel contesto finanziario e con la crescita del personale addetto allo studio dei dati è sempre più complessa la gestione di questo ramo di sviluppo in modo "tradizionale".

Precedentemente infatti, le politiche interne relative allo sviluppo della reportitica demandavano ai singoli team l'interno processo: essi, in autonomia, erano liberi di scegliere strumenti e dati su cui andare a costruire i propri report. L'idea alla base è nobile, tuttavia porta con se una serie di problematiche che vanno prese seriamente in considerazione:

1. Duplicazione e ridondanza nei dati: questo porta problematiche di inefficienza a livello di spazio e mantenimento dei dati
2. Difficoltà di condivisione tra team differenti: la condivisione tra team di ambiti differenti all'interno di uno stesso gruppo è cruciale, avendo spesso software differenti non era così immediato
3. Mancanza di dati certificati: potendo scegliere in autonomia le sorgenti dati da utilizzare il rischio di non avere dati certificati ed allineati con gli altri team è alto e questo porta rischi elevati durante l'analisi dei report stessi
4. Inefficienza sulle progettualità: lavorando ognuno per conto suo spesso ci si trovava nella condizione di rifare lavori già fatti spendendo risorse
5. Mancanza di standard e curva di apprendimento lenta: la grossa frammentazione tecnologica porta con se in primis una mancanza di standard a livello di

costruzione di repostistica ma anche un difficile adattamento per dipendenti che lavorano cross team dovendo imparare da zero un nuovo processo ad ogni spostamento

Per cercare di risolvere questi problemi è stato richiesta la realizzazione di un modello dati polifunzionale, centralizzato collegato ad un sub-set di sorgenti dati certificate capace di adempiere alle richieste di tutti gli utenti coinvolti. In un secondo momento gli utenti saranno completamente autonomi nella realizzazione dei report effettivi sugli ambiti reali sfruttando una piattaforma comune in cui la condivisione è semplice e immediata.

Il risultato è stato possibile raggiungerlo andando a progettare la soluzione in modalità self BI, il team di sviluppo dunque si è occupato in primis dell'analisi delle esigenze degli utenti andando a costruire il modello dati connettendo opportunamente sorgenti dati eterogenee e relazionandole in modo tale da rendere possibili i casi d'uso presentati dagli utenti business. In un secondo rilasciando e mantenendo il dataset dando modo agli utenti di realizzare i report di loro interesse sugli ambiti trattati.

Power BI

Il software scelto per la realizzazione della soluzione richiesta dal cliente è stato Power BI[5], piattaforma appartenente alla power platfolrm di Microsoft e leader nel settore della business intelligence. Power BI consente di andare ad aggregare, relazionare e modellare dati strutturati provenienti da origini dati eterogenee. Oltre che la semplice modellazione il software rende possibile l'ampliamento e arricchimento dei dati tramite misure e colonne calcolate che permettono di avere aggregazioni in tempo reale su porzioni di dati scelti.

Power BI accetta dati dai principali provider come: Orcle, SQL Server, MySQL e postgress. Oltre che da Database veri e propri è possibile andare a caricare nel modello file excel strutturati rendendo dunque ancora più immediato l'ampliamento del perimetro. Esistono due tipologie di connessioni verso le fonti alimentanti:

1. Direct Query: Questa modalità è la più leggera e spesso viene utilizzata per sfruttare sorgenti dati con Terabyte di dati, il modello non fa altro che andare a salvare internamente il tracciato della tabelle (le colonne) e informazioni sul typing. Successivamente ad ogni interazione dell'utente con il modello esegue a run-time una query al database di riferimento con il solo sub-set di informazioni richieste. Questo permette di evitare al modello di scaricare una

[5]https://powerbi.microsoft.com/

quantità di dati non gestibile e su cui non si potrebbero applicare aggregazioni senza valutare appositi filtri per ridurne la numerosità. Questa modalità porta con se un altro beneficio, cioè il fatto che il modello dati risulti sempre aggiornato in tempo reale rispetto ai database di riferimento

2. Import: Come suggerisce il nome questa modalità è l'esatto opposto di quella appena enunciata. Il modello scarica tutti i dati presenti nella tabella target salvandoseli internamente, questo ha come beneficio sicuramente il tempo di attesa per estrarre informazioni: a differenza della modalità Direct Query è già tutto salvato e indicizzato e non è dunque necessario attendere il tempo di una select. Come sempre però ci sono anche dei contro: il modello risaltante non è più aggiornato in tempo reale rispetto alla base dati, ma bensì l'informazione disponibile è ferma all'ultimo download. Un aggiornamento, che come detto comporta uno scarico massivo dei dati ha delle tempistiche differenti rispetto a salvarsi solo il tracciato.

La scelta della modalità di importazione di pende strettamente dalla tipologia di dati e dalle necessità specifiche, in questo caso specifico essendo il modello dati piuttosto complesso di è optato per una soluzione ibrida in cui le tabelle dei fatti pescano in direct query direttamente dal database i dati mentre le dimensioni, spesso anagrafiche, essendo nell'ordine di centinaia di righe, sfruttano la modalità import.

In definitiva Power BI è uno strumento semplice, ma al-contempo potente che permette di realizzare modelli dati su cui effettuare analisi. Naturalmente la parte fondamenta è quella di progettazione dell'architettura della soluzione per rendere coerente la struttura con le aspettative del cliente.

Power BI Service

Power BI service è la versione cloud del servizio di business intelligence di Microsoft, è pensato per essere fruibile da ogni dispositivo connesso ad internet che esso sia un pc o uno smartphone permettendo dunque duttilità e semplicità di utilizzo. PBI service permette agli utenti di visionare report precedentemente realizzati con referesh dei dati in tempo reale oppure costruire report in autonomia da zero.

Un punto fondamentale del servizio è la gestione delle cartelle di lavoro. Ogni gruppo di lavoro è organizzato con un workspace dedicato così da poter rendere fruibili i report costruiti ad altri colleghi senza dover fare operazioni complesse, inoltre per gli utenti è possibile andare a condividere esternamente, con utenti autorizzati, i loro lavori abbattendo le tempistiche e migliorando il knowledge sharing.

Il servizio offre moltissime altre funzionalità interessanti, una su tutte la costruzione di web-app vere e proprie, il loro scopo è quello di raggruppare grosse quantità

di report in un unico posto semplice da raggiungere. All'interno del progetto è presente un'app di "catalogo" condivisa tra tutti i gruppi di lavoro che permette di avere a disposizione una vasta gamma di report generali da consultare giornalmente nella quale automaticamente in base all'utenza mostra solo i dati di pertinenza di quello specifico utente.

Architettura del modello BI

Il progetto come anticipato ha come componente principale il software power BI sul quale sono stati sviluppati tutti i dataset da cui gli utenti hanno realizzato i loro report in cloud, sfruttando le potenzialità di power bi service descritte precedentemente. Tutta via l'architettura è leggermente più complessa di quanto raccontato sino ad ora, infatti power BI non è altro che un aggregatore di dati già presenti e gestiti da terze parti, la figura 4.1 cerca di mostrare come tutti gli elementi che entrano in gioco prendono parte alla soluzione messa in atto. Si noti

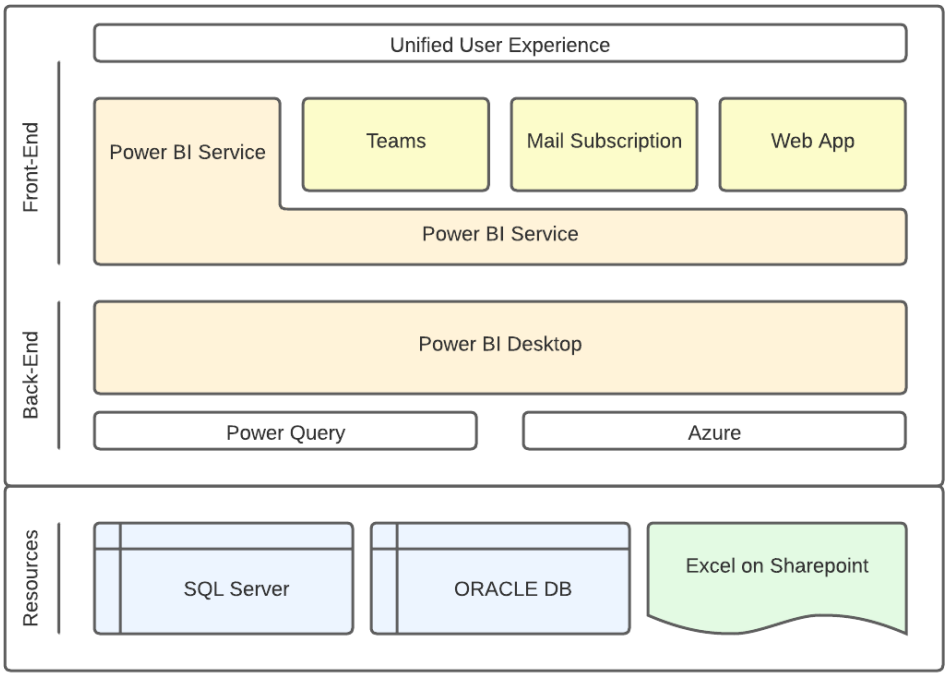

Figura 4.1: Rappresentazione grafica dell'architettura della soluzione di business intelligence sviluppata per il cliente

come l'architettura è suddivisa in tre sezioni principali, partendo dal basso verso l'alto:

1. Resources: Rappresenta l'insieme di servizi che espongono al modello dati le informazione che vengono successivamente modellate all'interno del software di business intelligence. Principalmente i dati che vengono utilizzati provengono da due database relazionali che espongono informazioni che vengono successivamente modellate nuovamente al fine di raggiungere il risultato atteso. Si noti come anche un elemento fondamentale sono i fogli excel esposti su sharepoint[6] essi infatti non contengono propriamente dati ma bensì classificazioni custom prodotte dagli utenti. Potendosi connettere direttamente a excel gestiti dagli utenti stessi è possibile dar loro la libertà di lavorare su specifiche riclassificazioni dei dati consolidati provenienti dai due database così da permettere di mantenere il più possibile la duttilità nella rappresentazione delle informazioni.

2. Back-End: Come già sottolineato più volte il back-end è gestito interamente da Power BI desktop sul quale vengono modellati i dataset su cui successivamente verranno realizzati i report. Non esiste un unico modello dati ma molteplici, in linea generale uno per ogni macro concetto su cui si vuole lavorare. Questa molteplicità di dataset si ripercuote anche sulla documentazione, infatti è importante per un'utente capire a quale mopdello dati è collegato il report di interesse. A Back-End è inoltre possibile aggiungere nuove tipologie di dati dette misure: esse permettono di ottenre KPI[7] aggregando informazioni presenti con differenti filtri applicati così da poter dare nuovo valore ai dati, queste misure sono scritte con un linguaggio chiamato DAX[8] che adotta un paradigma di programmazione di tipo funzionale.

3. Front-End: Quest'ultima sezione, quella a stretto contatto con l'utente, racchiude tutte le diverse vie in cui si può interagire con il modello dati, i report vengono sempre realizzati all'interno del servizio PBI service e successivamente possono essere esposti in differenti modi, Microsoft infatti permette di integrare agevolmente tutti i report presenti sul servizio cloud nelle varie applicazioni di office 365, da teams fino ai powepoint. Il lavoro degli utenti tuttavia può essere embeddato anche in webapplication custom permettendo di avere continuità nel modo di interagire con i dati anche al di fuori dei servizi Microsoft. In ultimo un altro modo per accedere alle informazioni presenti nel modello dati è tramite sottoscrizione di mail, anche questo servizio è fornito da PBI service e permette agli utenti di riceve direttamente per mail, con cadenza personalizzata, report già pre-impostati per monitorare i proprio portafogli di

[6]Servizio di file sharing di Microsoft

[7]Key point of intrest

[8]Data Analysis Expressions

investimento. Tali mail possono anche essere personalizzati per inviare una notifica per determinati trigger event, ad esempio quando un KPI scende o sale rispetto a limiti previsti.

Tutti questi elementi assieme permettono di ottenere una soluzione completa che soddisfa le esigenze richieste, tuttavia il progetto è ancora in sviluppo e i modelli dati in continua evoluzione sulla base delle richieste utenti. Attualmente il servizio viene utilizzato da circa trecento utenti provenienti da 6 paesi europei, i report costruiti invece ad oggi sono più di 800 con un tasso di crescita costante. Per quanto riguarda le numeriche più tecniche attualmente all'interno dei differenti dataset ci sono più di 100 tabelle per un complessivo di 1200 campi utilizzabili all'interno dei report. Ogni tre mesi viene tenuto un keynote interno per aggiornare gli utenti sullo stato avanzamento del progetto e presentare nuove idee e nuovi dati da poter utilizzare, inoltre questo momento viene sfruttato dagli utenti stessi per presentare loro alcuni report che hanno realizzato.

Complessità affrontate

La realizzazione di tale modello porta con se differenti sfide tecniche-organizzative, una su tutte è la gestione della documentazione dei campi e KPI[9] utilizzabili dagli utenti nei loro report. Nel tempo il modello dati è cresciuto moltissimo arrivando a raggiungere decine di tabelle contenenti centinaia di campi con significati differenti tra loro. Questo ha reso più ripida la curva di apprendimento per i nuovi utenti che si approcciavano alla costruzione di reportistica non sapendo rispetto alla loro esigenza quale fosse il miglior campo da utilizzare.

Nel tempo il massiccio utilizzo dello strumento ha portato con se un'altra esigenza, un utente prima di realizzarsi da zero un report secondo le sue necessità vorrebbe poterlo "ricercare" tra le decine prodotte da altri utenti e di conseguenza sfruttare un lavoro già svolto o, in alternativa, avere un punto di partenza con ridurre l'effrot di lavoro.

La prima soluzione volta a mitigare queste problematiche è stata la realizzazione di una base documentale su excel che rappresentasse e organizzasse tutte le informazioni necessaria agli utenti che a noi sviluppatori. Sono stati realizzati due differenti data dictionary:

1. Il pirmo rappresenta gli elementi contenuti nel modello dati, tutti i campi e KPI di ogni singola tabella rappresentano una entry nel nostro data Dictionary e sono corredati da una serie di informazione tecniche quali la provenienza

[9]Indicatore chiave di prestazione (Performance indicator)

del campo, il codice DAX[10] nel caso di misure e note aggiuntive di commento presenta una descrizione funzionale dettagliata che spiega cosa rappresenta quello specifico elemento e in che contesti ci si aspetta venga utilizzato. Questa descrizione funzionale risulta fondamentale per gli utenti per capire cosa rappresenta un determinato KPI.

2. Il secondo documento invece si concentra sulla descrizione dei report progettati e realizzati dagli utenti stessi, similmente a prima ogni report possiede una descrizione dettagliata di cosa rappresenta e quali esigenze riesce a soddisfare, anche in questo caso risulta fondamentale per gli utenti per riuscire ad avere più informazioni possibili sul lavoro esistente.

4.2.2 Obiettivo POC

L'obbiettivo della POC proposta e sviluppata è quello di riuscire a migliorare e rendere più efficiente l'interazione degli utenti con la documentazione presente sul progetto, cercando di dar loro uno strumento semplice ed intuitivo con cui interagire con linguaggio naturale ed ottene informazioni verticali su quanto richiesto. Il punto chiave è abbattere le tempistiche e mitigare la difficoltà di iterazione con la documentazione rendendola semplice anche per neofiti del progetto.

Avendo dunque una documentazione ricca da cui partire partire le prime fasi di analisi si sono concentrate maggiormente su come rappresentarla e come permettere agli utenti di interagire nel modo più semplice e naturale possibile.

4.3 Architettura e tecnologie utilizzate

La fase di analisi e progettazione della soluzione è stata la parte più ricca e complessa, infatti l'idea di base era quella si di fornire una soluzione completa, ma anche quella di cercare di contenere i costi di sviluppo e run del servizio. Il primo approccio è stato quello di valutare servizi auto-contenuti offerti da Azure per cercare di abbattere le tempistiche di sviluppo e tentare con un approccio semplicistico. Il primo impianto dunque è stato utilizzato sfruttando il servizio Azure cognitive search.

4.3.1 Azure cognitive search

Azure cognitive search è un servizio auto-contenuto all'interno della piattaforma Azure che permette di andare a fare ricerca avanzata su una base documentale

[10]Data Analysis Expressions, linguaggio per realizzare misure custom in PBI

precedentemente indicizzata. La versione di cognitive search presa in analisi è relativa al periodo gennaio - marzo 2023 in cui non vi era disponibile, nemmeno in preview, la possibilità di rappresentare i documenti come vettore densi sfruttando modelli come ADA.

La versione considerata di cognive search permetteva dunque di andare ad indicizzare semplicemente i documenti con un approccio simile a quello dettagliato alla sezione 2.1, in cui quindi le informazioni non sono altro che lunghi vettori sparsi strettamente legati ai termini e alle loro occorrenze ricadendo dunque nelle problematiche classiche di questa tipologia di rappresentazione.

Il servizio tuttavia aveva alcune caratteristiche peculiari come ad esempio la possibilità di aggiornare gli indici di ricerca all'aggiornamento della base documentale di partenza. La funzionalità descritta certamente avrebbe semplificato di molto il mantenimento del servizio, ma analizzando il caso specifico gli aggiornamenti della base documentale non sono così ripetuti nel tempo da rendere necessaria questa funzionalità.

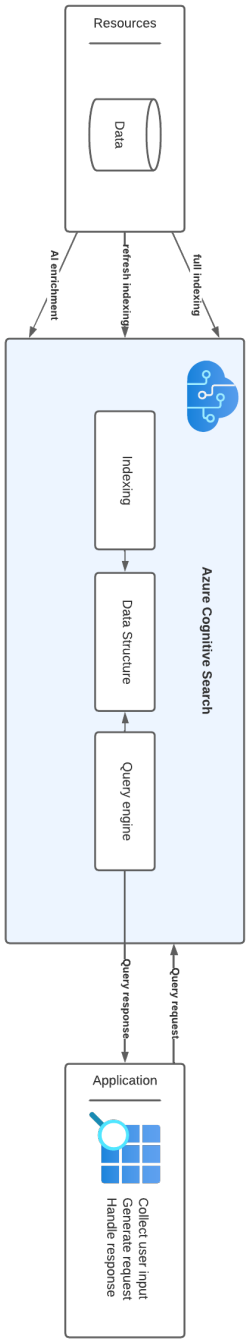

Figura 4.2: Rappresentazione grafica dell'architettura Azure cognitive serach utilizzata in fase sperimentale della soluzione

La soluzione messa in atto e rappresentata in figura 4.2 tuttavia non forniva il

grado di accuratezza e esperienza utente a cui puntavamo, essa infatti non faceva altro che restituire in output una "classifica" di righe del corpus che facevano maggiormente mach con la query utente lavorando con un approccio keyword based, dunque poco robusto ai sinonimi e a differenti modi di esprimere concetti simili. Altro lato negativo l'impossibilità di ricercare elementi in più lingue se non l'Inglese, stessa lingua della documentazione. Essendoci però molti team sparsi per i vari paesi europei fornire un'interazione nella lingua parlata avrebbe portato al servizio un valore aggiunto.

Per questi motivi lo sviluppo è proseguito in un altro senso focalizzandoci maggiormente sulla rappresentazione semantica degli elementi del corpus così da rendere la base dati interpellatile in differenti lingue e la ricerca più robusta ai sinonimi. Proprio in questo momento entrano in gioco i database vettoriali essi infatti sono cruciali per andare a salvare le informazioni del corpus in forma vettoriale e successivamente ricercarle sulla base della query utente. Punto cruciale era l'utilizzo della ricerca ibrida dato che le descrizioni all'interno del corpus hanno una serie di parole chiave specifiche del contesto di cui si sarebbe persa traccia nella rappresentazione semantica. Per questo motivo si è optato per integrare un database vettoriale che integrasse nativamente questa features, la scelta e ricaduta su weaviate. La scelta di questo specifico database vettoriale è stato prevalentemente per motivi di policy del cliente, infatti i dati interni possono essere gestiti solamente da software approvati con licenza, rendendo di fatto non utilizzabile la proposta di ricerca ibrida approfondita nella sezione 3.3.7. Tuttavia l'idea alla base della implementazione nativa resta fedele a quanto trattato sperimentalmente sul database vettoriale Qdrant.

Nello stesso periodo dello sviluppo è stato presentato e rilasciato chatGPT spostando moltissimo il focus sulle intelligenze artificiali generative con cui poter interagire in maniera attiva conversando in linguaggio naturale e ottenendo informazioni. Anche per questo motivo la direzione durante la fase di progettazione è andata sempre più verso un approccio "chatGPT like" rispetto ad una semplice ricerca per key-word.

L'architettura target per questa soluzione è rappresentata in figura 4.3, si noti come vengano utilizzati sia modelli generativi come GPT-4 che modelli di embedding come ADA, tutto questo orchestrato all'interno di Azure. Tutti i dati sono invece gestiti da un database vettoriale che ne permette la ricerca a livello semantico. La parte di iterazione con l'utente è modulare e si può adattare in base all'esigenza, essendo totalmente REST[11] l'architettura è semplice andare a modificare il front-end per rendere il sistema fruibile in differenti contesti. La

[11]REpresentational State Transfer

POC è stata costruita utilizzando Gradio per avere un'interfaccia web semplice ed intuitiva.

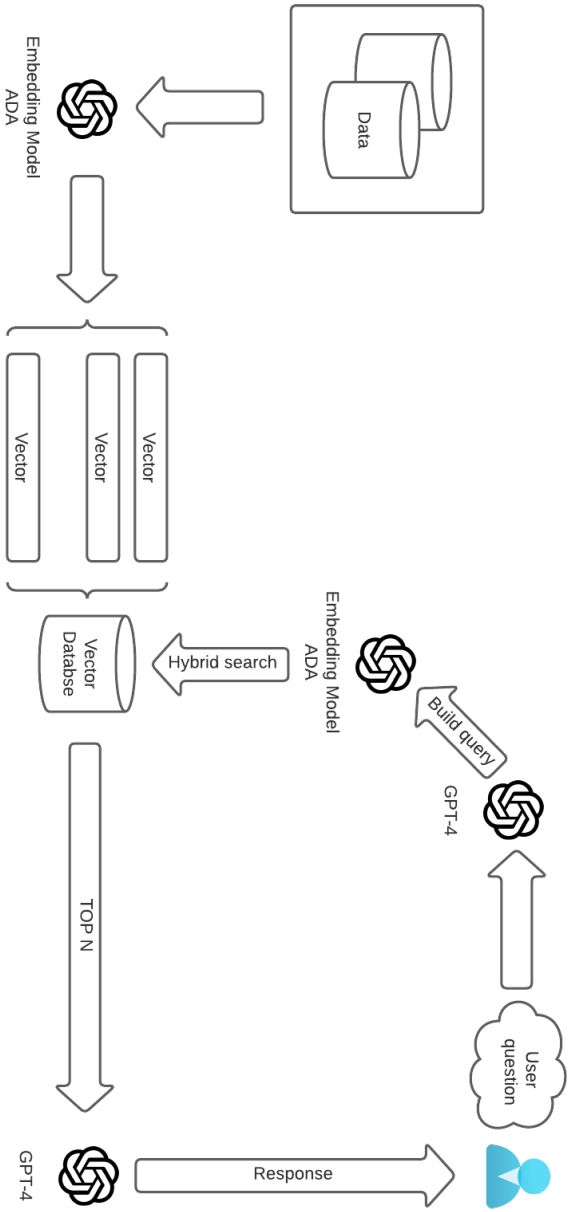

Figura 4.3: Rappresentazione grafica dell'architettura utilizzata per la realizzazione del proof of concept in oggetto

4.3.2 ChatGPT plugin

Per lo sviluppo della soluzione è stato utilizzato come punto di partenza il plugin chatGPT retrieval[12]

I plugin sono generalmente un'estensione di un applicativi o, come in questo caso, di modelli linguistici come ChatGPT. L'obiettivo è quello di consentire loro di accedere a informazioni aggiornate, eseguire calcoli o interagire con servizi di terze parti in risposta alla richiesta di un utente. Questo permette di sbloccare un'ampia gamma di potenziali casi d'uso e migliorano le capacità dei modelli linguistici, andando di fatto a superare il limite relativo alla conoscenza ferma e limitata al momento dell'addestramento.

Dunque è possibile andare a creare un plugin esponendo un'API tramite il proprio sito Web e fornendo un file manifest standardizzato che la descrive. ChatGPT grazie a questi file e consente ai modelli AI di effettuare chiamate all'API definita precedentemente dallo sviluppatore.

Un plugin generalmente è composto da:

- Un'API
- Uno schema API (formato OpenAPI JSON o YAML)
- Un manifest (file JSON) che definisce i meta-dati rilevanti per il plugin

In questo caso specifico parliamo di un plugin per ChatGPT che consente la ricerca semantica e il recupero di documenti scelti dall'utente. Permette quindi di ottenere i frammenti di documenti ordinati in base alla rilevanza rilevanti utilizzando come knowledge base le fonti dati scelte, come file, note o e-mail, ponendo domande o esprimendo esigenze in linguaggio naturale. Questo è esattamente ciò che serve nel caso specifico della POC, esporre dunque documenti agli utenti tramite ChatGPT utilizzando questo plugin.

Il plugin, come vedremo più in dettaglio nella sezione successiva, utilizza il modello di embedding di OpenAI text-embedding-ada-002 per generare i vettori densi di blocchi di documenti, successivamente sono archiviati e interrogarli utilizzando un database vettoriale sul back-end.

Essendo questa una soluzione open source e self-hosted, gli sviluppatori possono distribuire come meglio credono il servizio, nel nostro caso la scelta è virata su Azure. Il plugin di retrieval supporta diversi provider di database vettoriali, consentendo agli sviluppatori di scegliere quello preferito da un elenco.

[12] github.com/openai/chatgpt-retrieval-plugin

Un server FastAPI espone gli endpoint del plugin per l'upserting, l'interrogazione e l'eliminazione dei documenti. Gli utenti possono perfezionare i risultati della ricerca utilizzando filtri di meta-dati per fonte, data, autore o altri criteri. Il plugin può essere hostato su qualsiasi piattaforma cloud che supporti i contenitori Docker, come Fly.io, Heroku, Render o gli app container di Azure che alla fine abbiamo scelto. Per mantenere aggiornato il database vettoriale con i documenti più recenti, il plugin può elaborare e archiviare continuamente documenti da più origini dati, utilizzando i webhook in entrata per gli endpoint di upsert ed eliminazione. Vengono supportati strumenti come Zapier o Make possono aiutare a configurare i webhook in base a eventi o pianificazioni.

Il plugin per il retrieval è stato creato utilizzando FastAPI[13], un framework web per la creazione di API con Python. FastAPI consente uno sviluppo semplice.

Uno dei vantaggi derivanti dall'utilizzo di FastAPI è la generazione automatica della documentazione API interattiva con l'interfaccia utente Swagger. Quando l'API è in esecuzione localmente, l'interfaccia utente di Swagger può essere utilizzata per interagire con gli endpoint API, testarne la funzionalità e visualizzare i modelli di richiesta e risposta previsti.

Il plugin espone gli endpoint per l'upserting, l'interrogazione e l'eliminazione di documenti dal database vettoriale. Tutte le richieste e le risposte sono in formato JSON e richiedono un token valido come autorizzazione all'interno dell'intestazione.

4.3.3 ADA V2

Il modello scelto per ottenere gli embedding è stato ADA V2[14] servizio offerto sempre da Open-AI e presentato nel 2022 con l'obiettivo di rendere semplice e fruibile la rappresentazione vettoriale anche di contesti lunghi. ADA, essendo basato su transformer, non lavora con parole ma bensì con token, nello specifico questo modello utilizza il tokenizer cl100k_base. Il suo contesto rispetto alla precedente versione è quadruplicato arrivando ad accettare in input fino 8196 token permettendo più flessibilità nella gestione di lunghi documenti.

La rappresentazione che offre ADA è un vettore denso di 1536 dimensioni, un decremento considerevole rispetto ai suoi predecessori, questo permette di lavorare più agevolmente con i database vettoriali e salvare grandi quantità di embeddings senza avere bisogno di grandi quantità di spazio. Questi sono stati i motivi principale che hanno portato a scegliere questo modello.

[13] fastapi.tiangolo.com

[14] text-embedding-ada-002

All'interno dell'architettura si può notare che il modello viene utilizzato due volte, la prima volta con un'esecuzione one-shot permette di trasformare in embedding l'intera base documentale che nel caso della POC in oggetto sono i due excel di documentazione descritti alla sezione 4.2. Dunque ogni riga di documentazione viene trasformata in un vettore denso a 1536 dimensione che viene successivamente salvato nell'apposita collezione sul vector database.

Il secondo utilizzo del modello ADA avviene in fase di query planning, infatti permette di andare a codificare in embedding l'output del pre-process della query utente espressa in linguaggio naturale. Questa operazione è fondamentale per rendere confrontabile la query e i dati precedentemente salvati all'interno del database utilizzando la cosine similarity.

4.3.4 GPT-4

Il large language model GPT-4[15] è stato cruciale per lo sviluppo della proposta, esso infatti ha permesso di dare espressività e potere generativo al risultato finale, inoltre rispetto al suo predecessore GPT-3.5 [13] permette di ricevere in input fino a 32.000 Token permettendo di mantenere uno storico della conversazione più lungo.

GPT-4 è stato presentato al pubblico nel marzo del 2023 come naturale evoluzione delle suoi predecessori, Secondo alcune fonti, GPT-4 arriverebbe ad avere fino a 1,7 trilioni di parametri[16]. Ciò lo renderebbe 1000 volte più grande di GPT-3, che aveva rispettivamente 175 miliardi di parametri. Tuttavia, OpenAI ha rifiutato di rivelare il numero esatto di parametri, quindi non è confermato ed essendo un modello closed source non si ha traccia. Alcune altre fonti suggeriscono che GPT-4 abbia 100 trilioni di parametri o 1 trilione di parametri. Pertanto, il numero in GPT-4 è ancora incerto e può variare a seconda della fonte.

Altra caratteristica peculiare di questo nuovo modello è l'adozione all'interno dell'architettura di Transformer XL [14] in grado di gestire un alto numero di Token[17] in input rendendolo ancor più potente. Open-AI con GPT-4 si è concentrata anche su un'altra feature, infatti il LLM è multi-modale, questo gli permette di lavorare non solo con testo, ma anche con immagini rendendo possibili task di classificazione o image recognition. Il modello riesce anche, data un immagine, a fornire una descrizione più o meno dettagliata di ciò che "vede". Questa funzionalità apre diversi scenari e possibilità per la POC discussa, questo punto verrà trattato nella sezione 4.5.

[15] openai.com/research/gpt-4

[16] Un parametro è un peso che la rete neurale apprende in fase di addestramento

[17] è una porzione di parola ed è ciò che accettano in input i LLM

L'utilizzo in preview di questo modello è stato possibile sfruttando le API Microsoft per Open-AI. Questo perché essendo Cluster Reply partner gold di Microsoft ha accesso ad una serie di servizi in anticipo rispetto al mercato pubblico, questo ha reso possibile sperimentare e realizzare questa POC in esclusiva su questa tecnologia.

Prompting

Le sfide nell'utilizzo di questo modello sono state principalmente a livello di prompting, infatti l'obiettivo era quello di evitare che le risposte divagassero rispetto ai risultati trovati all'interno del vector database per ottenere un risultato appropriato l'approccio è stato per tentativi successivi tenendo a mente sempre le best practices quali:

- Usare istruzioni chiare e specifiche cercando di evitare le negazioni
- Sfruttare segni di punteggiatura e simboli per arricchire le richieste
- Forzare il modello ad eseguire una serie di step per ottenere l'output desiderato

Come si nota dall'architettura riportata in figura 4.3 il modello GPT-4 viene interpellato 2 volte per due scopi differenti nel primo caso l'obiettivo, espresso sotto forma di promt è: Analizzare la conversazione tra assistente (A:) e utente (Q:). Lo scopo ultimo è andare a modellare la struttura dell'oggetto JSON che successivamente verrà utilizzato per effettuare la query verso il vector database. La seconda richiesta fatta all'API di GPT-4 è a valle della ricerca effettuata sul DB in cui una volta restituiti N risultati, quelli con similarità maggiore, il LLM si occupa di generare un output sulla base dei risultati ricevuti e, se c'è, sul contesto della conversazione presente.

Entrambi i promt costruiti per utilizzare il modello presentano una struttura a step in cui come prima cosa viene spiegato al modello in linguaggio naturale qual è il goal e che cosa sta "interpretando" il modello, successivamente invece come deve comportarsi per raggiungere l'obiettivo espresso. Nelle fasi successive di sviluppo sempre grazie al promt sono stati aggiunti due elementi, il primo grazie all'utenza in sessione nel servizio salvare il nome dell'interlocutore, così da avere un'iterazione più diretta. La seconda invece è stata inserire la possibilità di modificare il tono della conversazione aggiungendo un parametro "tone" all'intero promt così da dare maggiore flessibilità e naturalezza al sistema.

4.3.5 Azure

Azure è la piattaforma cloud pubblica di Microsoft, offre una vasta gamma di servizi tra cui risorse di elaborazione, archiviazione, memorizzazione, trasmissione

dati e interconnessione di reti, analisi, intelligence, apprendimento automatico, sicurezza e gestione delle identità, monitoraggio e gestione, nonché servizi per lo sviluppo di applicazioni.

Essendo l'azienda per cui è stata sviluppata la POC partner gold di Microsoft la scelta per questo servizio è stato immediato, l'utilizzo del servizio cloud è stato in due direzioni, sia come Iaas[18] che come PaaS[19].

Le principali componenti infrastrutturali utilizzate sono state gli app container in cui sono stati caricate le immagini docker dei vari servizi come il plugin chatGPT e il vector database Weaviate. Azure essendo una piattaforma cloud permette di gestire agevolamento il carico di lavoro andando a scalare le risorse per permettere al servizio di risultare performante e disponibile.

I servizi utilizzati sono principalmente della suite Open-AI che hanno permesso l'utilizzo dei modelli ADA e GPT-4 dettagliati nelle sezioni precedenti.

4.3.6 Gradio

Gradio è una libreria open-source che nasce per semplificare la creazione di interfacce utente web per python, questo consente agli sviluppatori di creare facilmente applicazioni interattive per la distribuzione dei propri servizi. Le caratteristiche principali di Gradio sono:

- Creazione dell'interfaccia utente: Una volta che hai il tuo modello, puoi utilizzare Gradio per creare un'interfaccia utente per esso. Puoi farlo definendo gli input e gli output dell'interfaccia utente, specificando i tipi di input accettati (come immagini, testo, audio, etc.) e i tipi di output che il tuo modello restituirà.

- Interfaccia utente: Gradio offre diverse funzionalità per realizzare l'interfaccia utente. Si possono aggiungere widget interattivi, impostare limiti sulla dimensione dei file, definire etichette per i campi di input, e molto altro ancora con il fine di migliorare l'esperienza di utilizzo per l'utente finale.

- Applicazione web: Una volta che si è configurata l'interfaccia utente, è possibile creare un'applicazione web che permette agli utenti di interagire con il modello sottostante, in questo caso specifico con il servizio di retrieval di informazioni dei documenti presenti nel vector database.

[18]Infrastructure as a Service

[19]Platform as a Service

- Esecuzione dell'applicazione: La web application è eseguibile dal proprio ambiente locale o all'interno di un server, nel caso specifico è stata puntualmente dockerizzata e inserita su Azure grazie ad un'app container. Gradio offre strumenti per avviare facilmente il server e gestire l'interfaccia utente.
- Monitoraggio e aggiornamenti: Gradio offre la possibilità di monitorare l'utilizzo dell'applicazione e, se necessario, aggiornare il modello sottostante senza dover apportare modifiche significative all'interfaccia utente.

La scelta di utilizzare gradio è stata fatta per abbattere le tempistiche e i costi di sviluppo data la sua semplicità di configurazione, nella sezione 4.5.1 si discuterà di quali altre possibili strade si sarebbero potute percorrere o si percorreranno.

4.4 Risultati ottenuti

Il risultato del proof of concept è quello desiderato, l'intero servizio è raggiungibile via web tramite utenza registrata e consente all'utente di scrivere messaggi in linguaggio naturale con il sistema chiedendo informazioni verticali sulla base documentale fornita. Come si può notare dall'immagine 4.4 l'utente può decidere inoltre di ri-generare una risposta che non lo ha soddisfatto oppure andare a pulire lo storico della conversazione evitando dunque di mentente il contesto precedente come base per le successive generazioni.

Il servizio da la possibilità all'utente di porre sia domande specifiche che generiche verticali sui documenti di input forniti al modello, nei successivi esempi la knowledge base salvata come vettore denso all'interno del database vettoriale è il documento di report dictionary dettagliato alla sezione 4.2. Nonostante l'intera base documentale presenta descrizioni in inglese le interazioni sono state eseguite in lingue differenti per mostrare le capacita multilinguali della soluzione.

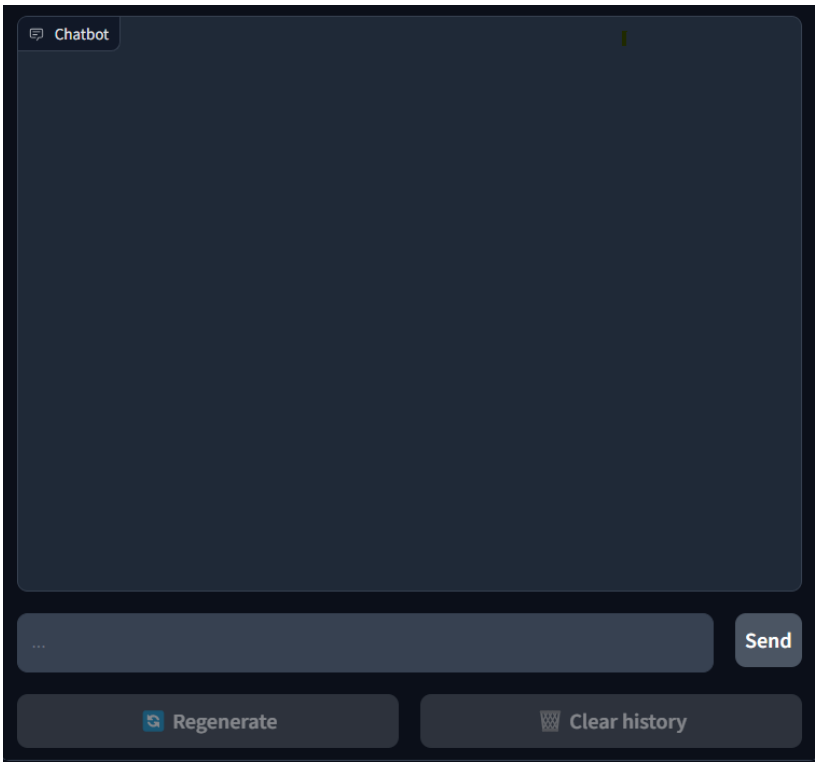

Figura 4.4: Screenshot del front end della POC, web-application sviluppata con Gradio

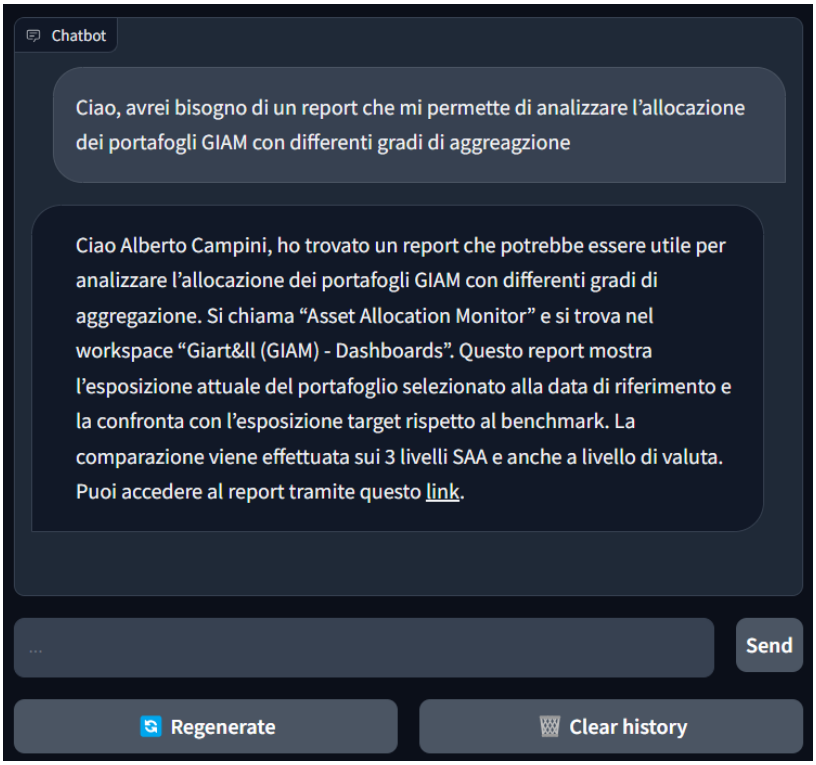

Figura 4.5: Esempio di conversazione in italiano con il chatbot cercando report specifico

Come si nota dalla figura 4.5 il modello come prima cosa identifica correttamente l'utente con cui sta interagendo grazie all'utenza con ci si è autenticati, successivamente confrontando la domanda posta con la KB cablata all'interno del vector database decide qual è o quali sono i punti della collezione più simili alla domanda posta e grazie al potere generativo di GPT-4 comunica all'utente la risposta puntuale. Analizzando l'output risalta la capacità di generare collegamenti iper-testuali che garantiscono immediatezza all'utente nel consultare il report proposto dal modello.

Come detto all'inizio della sezione uno degli obiettivi era quello di avere un servizio chatGPT like, dunque che permettesse all'utente di effettuare una vera e propria conversazione andando a chiedere via via informazioni più specifiche o anche differenti mantenendo sempre il contesto del dialogo. In figura 4.6 è possibile notare come proseguendo la conversazione e chiedendo un'alternativa il modello risponde con un report differente e spiega perché, per il modello, quello fornito sia un'alternativa valida alla prima riposta.

Da questa semplice interazione si nota come il contesto del dialogo lavori correttamente, la scelta di chiedere "un'alternativa" non è casuale, infatti, per

riuscire in questo intento il modello deve per forza riuscire ad utilizzare bene sia la mia precedente domanda che la sua successiva riproposta dato che per fornire in output un report atteso deve utilizzare tutte le informazioni nel suo contesto.

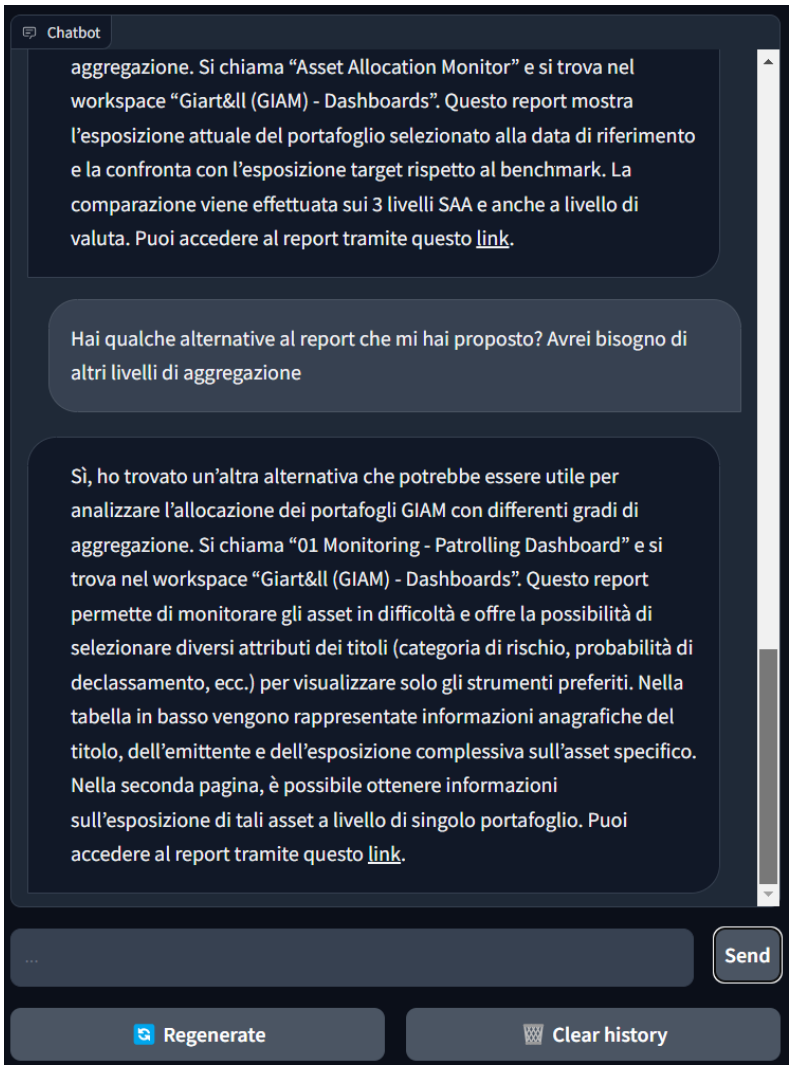

Figura 4.6: Esempio di molteplici interazioni con il chat-bot sulla base della storia della conversazione

Come successiva sperimentazioni è stata posta una domanda in spagnolo, simulando un utente proveniente dall’estero per verificare le capacità multi-linguali. Inoltre in questa conversazione rispetto alla precedete è stata posta una domanda più generica in cui ci si aspetta una serie di alternative relative all’analisi delle transazioni in differenti contesti.

Come si nota dalla figura 4.7 il modello, ponendosi in lingua spagnola, restituisce tre alternative valide rispetto alla richiesta posta. Il numero tre non è casuale, infatti i risultati della query eseguita sul vector database è una TOP-N dove N è uguale a 3. La scelta nasce dal trade-off tra potere di generalizzazione e costi del servizio. Tanto più input viene fornito a GPT-4 tanto più costerà la generazione dell'output.

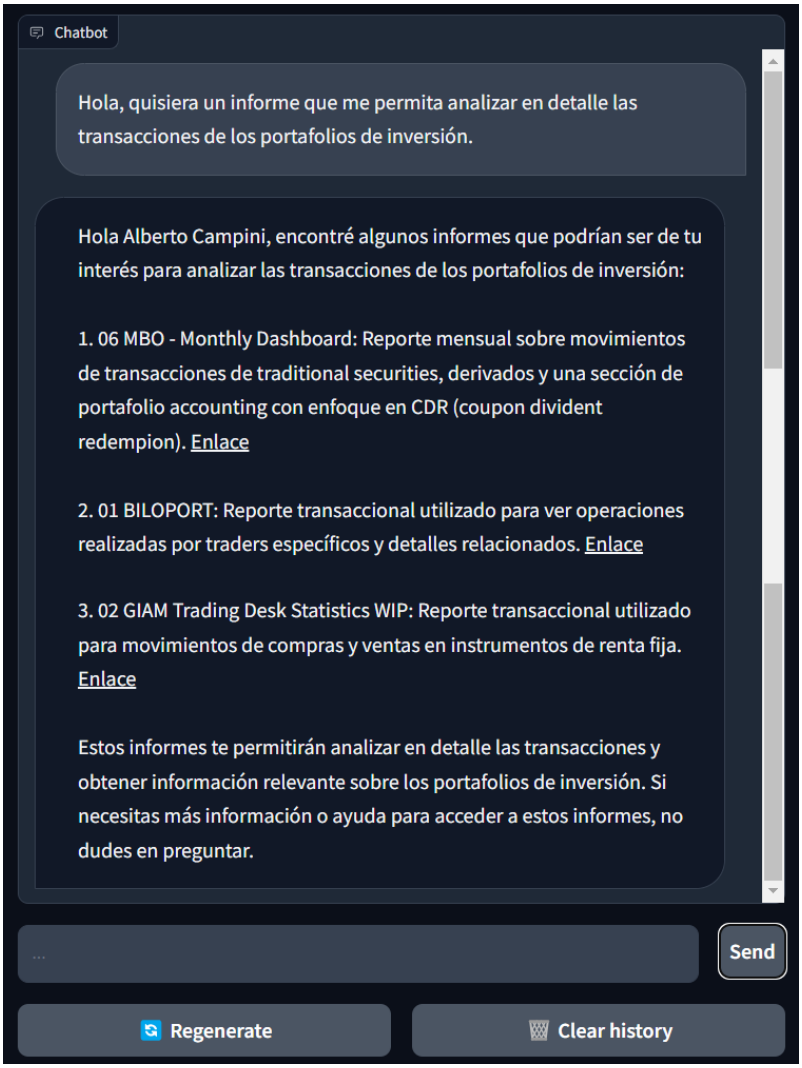

Figura 4.7: Esempio di interazione in spagnolo simulando un'interazione multi linguale

Limiti riscontrati

Nonostante l'applicativo abbia raggiunto le aspettative vi sono alcuni limiti attualmente presenti e strettamente legati alle tecnologia utilizzate uno su tutti è il contesto limitato, infatti se una conversazione si protrae a lungo il modello non riesce a valutare l'intero contesto presente, ma solamente la porzione che rientra nel numero massimo di token in input, questo è strettamente legato alla versione di GPT-4 scelta, infatti possiamo avere come limite 8.000 Token o 32.000. Naturalmente la scelta impatta anche i costi ad utilizzo del servizio. Il rischio dunque è che duramente una conversazione particolarmente lunga il modello ri-proponga risposte già fornite precedentemente di fatto andando a ripetersi. Tuttavia per il tipo di utilizzo per cui è stata pensata questa POC le conversazioni non dovrebbero prostrasi per più di 3-4 scambi di messaggi consentendo di rimenare nei limiti del contesto disponibile.

Un altro limite riscontrato, anch'esso portato dalla tecnologia scelta è l'incapacità del modello nel dire "non lo so" infatti una caratteristica dei modelli generativi come GPT è quella di non essere in grado in molti casi di dire semplicemente non lo so, infatti, salvo casi particolari in cui gli sviluppatori bloccano a monte il modello, l'output generato sarà un qualcosa di poco preciso o tal volta incorretto piuttosto che un feedback all'utente di mancanza di conoscenza. Attraverso il lavoro di promting questo problema è stato mitigato il più possibile cercando in primis di non farlo divagare rispetto ai risultati forniti dal database vettoriale e successivamente cercando di evitare per lo più risposte inventate o probabilmente fuorvianti.

4.5 Conclusioni e possibili sviluppi futuri

In conclusione lo sviluppo di questa POC ha dimostrato come tecnologie innovative come i vector database possono essere calate in differenti ambiti applicativi e progettuali andando a rendere possibile la costruzione di soluzioni ad-hoc per adempiere a richieste specifiche. L'utilizzo di modelli complessi come ADA per l'embedding e GPT-4 per la generazione ha permesso di toccare con mano la potenzialità della rappresentazione contestuale in vettori densi dell'informazione affrontata nella prima parte della trattazione, rendendo possibile il superamento dei limiti di lingua, sinonimia e ambiguità.

4.5.1 Interazione con l'utente

Come spiegato brevemente nella sezione 4.3 l'architettura proposta essendo REST permette di integrare front-end differenti per interagire con il modello. PEr permettere agli utenti di avere un'esperienza ancora più inversiva e coerente con il loro workflow si potrebbe valutare di sostituire il componente Gradio con un'alternativa. Ci sono differenti soluzioni ma a mio parere le due più interessanti da valutare sono:

- Copilot: Microsoft dopo l'investimento da 10 miliardi in Open AI ha lavorato duramente per integrare i servizi di intelligenza artificiale all'interno dei suoi prodotti, nell'ultimo periodo ha rilasciato in beta il copilot per office 365 esponendo anche librerie per sviluppare plugin per esso. Una possibilità sicuramente affascinante sarebbe quella di collegare al copilot di teams le API sviluppate per andare a fare retrieval sulla documentazione e demandare completamente al copilot l'interazione con l'utente. Questo permetterebbe di estendere un servizio già di per se completo con questa capacità in più così da permettere agli utenti di avere continuità di utilizzo durante le loro giornate lavorative. Attualmente il copilot non è disponibile per tutti e la beta è chiusa dunque questa evolutiva si potrà valutare non appena ci sarà un quadro più generale della disponibilità del servizio e delle effettive funzionalità.

- Bot Teams: Rimanendo sempre sul tema di esperienza utente e integrazione con il work flow, un'altra idea sarebbe quella di sviluppare un bot teams con le medesime capacità di Gradio, ma direttamente utilizzabile tramite client desktop. Le potenzialità tuttavia rimarrebbero invariate l'unica differenza tangibile sarebbe come e da dove intereagire con il sistema

Per quanto riguarda le possibili migliorie su questo ambito i feedback del cliente saranno fondamentali per capire, eventualmente, quale direzione intraprendere.

4.5.2 Image description

Utilizzando come LLM GPT-4 sarebbe possibile valutare l'utilizzo delle sue caratteristiche multi-modale per andare ad arricchire la KB del modello. Ipotizzando infatti di avere un nuovo report di cui si vuole generare una descrizione invece di andare ad inserire manualmente tale informazione sarebbe possibile dare in input al modello uno o più screenshot relativi al report e tramite promting chiedere una descrizione di che cosa "vede" andando ad automatizzare il processo di catalogazione dei report utente.

Questa possibile evolutiva non è banale come sembra infatti non è detto che il modello riesca ad essere sufficientemente preciso e sopratutto non essendo dato fatto un fine-tuning sull'ambito specifico potrebbe non avere a disposizione le informazioni necessarie a descrivere cosa gli viene dato in input. Questa sperimentazione non è stata esplorata per limitazione tecniche infatti nel periodo di sviluppo della POC le API in preview non permettevano di fornire in input immagini al modello generativo.

Bibliografia

[1] T. H. Nelson. «Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate». In: *Proceedings of the 1965 20th National Conference*. ACM '65. Cleveland, Ohio, USA: Association for Computing Machinery, 1965, pp. 84–100. ISBN: 9781450374958. DOI: 10.1145/800197.806036. URL: https://doi.org/10.1145/800197.806036 (cit. a p. 7).

[2] Daniel Jurafsky James H. Martin. In: *Speech and Language Processing*. 2000. DOI: https://web.stanford.edu/~jurafsky/slp3/ (cit. alle pp. 10–13, 18).

[3] S. F. Dierk. «The SMART retrieval system: Experiments in automatic document processing — Gerard Salton, Ed. (Englewood Cliffs, N.J.: Prentice-Hall, 1971, 556 pp., 15.00)». In: *IEEE Transactions on Professional Communication* PC-15.1 (1972), pp. 17–17. DOI: 10.1109/TPC.1972.6591971 (cit. a p. 12).

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin. «Attention Is All You Need». In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. a p. 16).

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee e Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). Esempi e spunti di analisi utilizzati per la trattazione della sezione sulla rappresentazione dell'informazione. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. a p. 16).

[6] Sung-Hyuk Cha. «Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions». In: 2007. URL: https://api.semanticscholar.org/CorpusID:15506682 (cit. a p. 17).

[7] Ravish Bhagdev, Sam Chapman, Fabio Ciravegna, Vitaveska Lanfranchi e Daniela Petrelli. «Hybrid Search: Effectively Combining Keywords and Semantic Searches». In: *The Semantic Web: Research and Applications*. A cura di Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann e Manolis Koubarakis.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 554–568. ISBN: 978-3-540-68234-9 (cit. a p. 34).

[8] Erica Mourão, Marcos Kalinowski, Leonardo Murta, Emilia Mendes e Claes Wohlin. «Investigating the Use of a Hybrid Search Strategy for Systematic Reviews». In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2017, pp. 193–198. DOI: 10.1109/ESEM.2017.30 (cit. a p. 34).

[9] Design proposto per sperimentare la ricerca ibrida dal creatore di Qdrant. URL: https://qdrant.tech/articles_data/hybrid-search/experiments-design.png (cit. a p. 34).

[10] Stephen Robertson. «Understanding Inverse Document Frequency: On Theoretical Arguments for IDF». In: *Journal of Documentation - J DOC* 60 (ott. 2004), pp. 503–520. DOI: 10.1108/00220410410560582 (cit. a p. 37).

[11] Stephen Robertson e Hugo Zaragoza. «The Probabilistic Relevance Framework: BM25 and Beyond». In: *Found. Trends Inf. Retr.* 3.4 (apr. 2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: https://doi.org/10.1561/1500000019 (cit. a p. 39).

[12] Rodger Benham e J. Shane Culpepper. «Risk-Reward Trade-Offs in Rank Fusion». In: *Proceedings of the 22nd Australasian Document Computing Symposium*. ADCS '17. Brisbane, QLD, Australia: Association for Computing Machinery, 2017. ISBN: 9781450363914. DOI: 10.1145/3166072.3166084. URL: https://doi.org/10.1145/3166072.3166084 (cit. a p. 41).

[13] Tom B. Brown et al. «Language Models are Few-Shot Learners». In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165 (cit. a p. 71).

[14] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le e Ruslan Salakhutdinov. «Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context». In: *CoRR* abs/1901.02860 (2019). arXiv: 1901.02860. URL: http://arxiv.org/abs/1901.02860 (cit. a p. 71).