Università della Calabria
Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Tesi di Laurea

GAEN: Generative AI for Enhanced Narrative - Rilevamento di Spoiler Mediante Modelli Generativi Avanzati

Relatori:
Prof. Gianluigi Greco
Prof. Kazumi Saito

Candidato:
Alessandro Fazio
Matricola 242422

Anno Accademico 2024/2025

Indice

Capitolo 1

Introduzione

1.1 Contesto

Nell'era digitale, l'accesso immediato a contenuti multimediali ha trasformato radicalmente il modo in cui fruiamo di film, serie TV, libri e videogiochi. Tuttavia, questa abbondanza di informazioni porta con sé una sfida: la crescente esposizione a spoiler. Gli spoiler, rivelando anticipatamente elementi cruciali della trama, possono compromettere l'esperienza di fruizione, generando frustrazione e delusione negli utenti.

Il problema degli spoiler è particolarmente rilevante in contesti online, dove le discussioni e le recensioni proliferano su forum, social media e piattaforme di streaming. La mancanza di strumenti efficaci per il rilevamento automatico di spoiler rende difficile per gli utenti proteggersi da tali rivelazioni indesiderate.

Questa tesi si propone di esplorare l'utilizzo di Large Language Models (LLM) per affrontare la sfida del rilevamento di spoiler. L'obiettivo è sviluppare un sistema in grado di identificare automaticamente gli spoiler in testi di varia natura, sfruttando le capacità di comprensione del linguaggio naturale degli LLM.

1.2 Motivazioni

La motivazione principale di questa ricerca risiede nella volontà di contribuire allo sviluppo di strumenti che migliorino l'esperienza di fruizione dei contenuti multimediali. A tale scopo, è stato realizzato un sistema di rilevamento di spoiler integrato in una estensione per browser, che offre agli utenti un controllo maggiore sulla propria esposizione agli spoiler.

Questa ricerca intende esplorare il potenziale degli LLM in un'applicazione pratica e rilevante, provando a contribuire alla comprensione delle loro capacità e limitazioni nel contesto del rilevamento di informazioni specifiche in testi complessi.

1.3 Obiettivi

Gli obiettivi di questa tesi sono i seguenti:

- Realizzare un sistema di rilevamento di spoiler basato su Large Language Models.
- Valutare l'efficacia del sistema sviluppato nel rilevamento di spoiler in testi di varia natura.
- Integrare il sistema in un'estensione per browser e valutarne l'utilità e l'usabilità.

1.4 Struttura della tesi

Il resto della tesi è organizzato come segue:

- Nel Capitolo 2 vengono presentati i concetti e le tecnologie alla base della ricerca, con particolare attenzione ai Large Language Models.
- Nel Capitolo 3 vengono esaminati i lavori correlati, con un focus sulle ricerche relative al rilevamento di spoiler.
- Nel Capitolo 4 viene descritto il sistema di rilevamento di spoiler sviluppato, con particolare attenzione alla progettazione e all'implementazione.
- Nel Capitolo 5 vengono presentati i risultati dell'analisi sperimentale condotta per valutare l'efficacia del sistema.
- Nel Capitolo 6 vengono riassunti i risultati ottenuti e vengono discusse le possibili direzioni future di ricerca.

Capitolo 2

Background

2.1 Approcci al problema

Di seguito vengono presentati alcuni approcci al problema di rilevazione di spoiler in un testo.

2.1.1 Approccio basato su regole

Un approccio molto semplice consiste nell'identificare alcune parole chiave che sono tipicamente presenti in un testo che contiene spoiler. Ad esempio, le parole "spoiler" e "spoiler alert" sono spesso utilizzate per avvertire il lettore che il testo successivo contiene informazioni che potrebbero rovinargli la visione di un film o la lettura di un libro. Altri esempi di parole chiave potrebbero essere i nomi di personaggi o luoghi chiave della trama. Questo approccio è molto semplice e può essere implementato con poche righe di codice, ma ha il difetto di essere molto limitato e di non essere in grado di rilevare spoiler più sottili o nascosti.

Questo approccio è inoltre soggetto a molti falsi positivi, ovvero a casi in cui il testo viene erroneamente classificato come contenente spoiler quando in realtà non lo contiene. Ad esempio, una recensione di un film potrebbe contenere il nome di un personaggio chiave della trama senza rivelare informazioni cruciali sulla storia. In questo caso, il testo non dovrebbe essere classificato come contenente spoiler, ma l'approccio basato su regole potrebbe erroneamente classificarlo come contenente spoiler.

Oltre ai falsi positivi, questo approccio è anche soggetto a falsi negativi, ovvero a casi in cui il testo contiene spoiler ma non viene classificato come tale. Ad esempio, una recensione di un film potrebbe contenere informazioni cruciali sulla trama senza utilizzare le parole chiave tipicamente associate

ai spoiler, o in casi anche più semplificati, potrebbe essere scritta in una lingua diversa da quella in cui sono state definite le parole chiave, o potrebbe contenere errori di battitura o di ortografia che impediscono al sistema di riconoscere le parole chiave.

Per questi motivi, l'approccio basato su regole non promette risultati soddisfacenti e si è deciso di non adottarlo per questo progetto.

2.1.2 Approccio basato su machine learning

Un approccio più sofisticato e promettente è quello basato su machine learning. In questo approccio, si addestra un modello di machine learning su un insieme di dati di addestramento etichettati, ovvero un insieme di testi già classificati come contenenti spoiler o non contenenti spoiler. Il modello di machine learning impara a riconoscere i pattern nei dati di addestramento e a classificare i testi in base a tali pattern. Una volta addestrato, il modello può essere utilizzato per classificare nuovi testi come contenenti spoiler o non contenenti spoiler.

Questo approccio ha il vantaggio di essere molto più flessibile e potente rispetto all'approccio basato su regole, in quanto il modello di machine learning è in grado di riconoscere pattern complessi e sottili nei dati di addestramento e di generalizzare tali pattern ai nuovi dati. Inoltre, il modello di machine learning è in grado di adattarsi automaticamente ai nuovi dati, senza la necessità di modificare manualmente le regole o i parametri del sistema.

Tuttavia, l'approccio basato su machine learning ha anche alcuni svantaggi. In primo luogo, richiede un insieme di dati di addestramento etichettati, che possono essere costosi e laboriosi da ottenere.

Il secondo svantaggio è che il modello di machine learning è difficile da realizzare e richiede competenze avanzate per la sua implementazione. Inoltre, il modello deve essere addestrato su un insieme di dati di addestramento rappresentativo e bilanciato, altrimenti potrebbe essere soggetto a overfitting o a underfitting.

Generalizzando, l'approccio basato su machine learning è più complesso e richiede più risorse rispetto all'approccio basato su regole, ma promette risultati migliori e più accurati.

2.2 Approccio proposto

Per affrontare il problema della rilevazione di spoiler in un testo, si propone di utilizzare un approccio basato su machine learning.

L'intuizione alla base di questo approccio è che i LLM (Large Language Model) sono in grado di catturare le relazioni semantiche e sintattiche tra le parole e i concetti all'interno di un testo, e quindi di riconoscere i pattern che caratterizzano i testi contenenti spoiler.

In particolare, si propone di utilizzare un modello di LLM pre-addestrato con lo scopo di estrarre lo spoiler da un testo piuttosto che classificare l'intero testo come contenente spoiler o non contenente spoiler, nel caso in cui il testo contenga spoiler.

Utilizzare un modello di LLM pre-addestrato ha il vantaggio di non richiedere un insieme di dati di addestramento etichettati, in quanto il modello è già stato addestrato su un vasto insieme di dati e ha imparato a riconoscere i pattern nei testi in modo automatico e generale.

A prescindere dal modello di LLM utilizzato, si propone di utilizzare un approccio basato su RAG (Retrieval-Augmented Generation) per arricchire il testo con informazioni realitive al contesto in cui è stato scritto. Ciò permette di migliorare la qualità delle predizioni del modello di LLM, in quanto il contesto può influenzare il significato delle parole e dei concetti all'interno del testo.

La difficolta principale di questo approccio è che i modelli di LLM sono molto complessi e richiedono risorse computazionali e di memoria considerevoli per essere addestrati e utilizzati. Inoltre, i modelli di LLM sono soggetti a fenomeni di allucinazione e di bias, che possono influenzare le predizioni del modello e ridurne l'accuratezza.

Tuttavia, nonostante queste difficoltà, gli LLM si sono dimostrati molto efficaci in una varietà di task di NLP (Natural Language Processing) e promettono risultati soddisfacenti per il problema della rilevazione di spoiler in un testo.

I dettagli dell'approccio proposto verranno discussi nei seguenti capitoli.

Capitolo 3

Lavori correlati

3.1 Contesto della Ricerca

In questo capitolo verranno presentati alcuni lavori correlati all'oggetto di questa tesi. In particolare saranno introdotti alcuni concetti chiave e verranno discussi in modo da poter contestualizzare meglio il lavoro svolto.

3.2 LLM, modelli di linguaggio

I Large Language Models (LLM) rappresentano una classe di modelli di deep learning che hanno dimostrato capacità straordinarie nel comprendere e generare testo in linguaggio naturale. Questi modelli sono addestrati su enormi quantità di dati testuali e sono in grado di apprendere complesse relazioni semantiche e sintattiche tra le parole.

Nel contesto del rilevamento di spoiler, gli LLM offrono un potenziale significativo grazie alla loro capacità di comprendere il contesto e il significato delle parole. A differenza dei modelli basati su regole o su approcci di machine learning tradizionali, gli LLM possono catturare sfumature semantiche e identificare spoiler anche in testi complessi e ambigui.

I primi modelli di LLM utilizzavano una architettura del tipo *encoder-decoder* per generare testo in linguaggio naturale. L'encoder è responsabile di trasformare il testo in input in una rappresentazione vettoriale, mentre il decoder è responsabile di generare il testo in output.

Con l'introduzione del meccanismo di attenzione, i modelli LLM sono diventati sempre più potenti e sofisticati. Il meccanismo di attenzione permette al modello di estrarre il contesto delle parole nel testo, migliorando notevolemente le performance in una varietà di task.

L'attenzione è stata successivamente estesa a modelli completamente *attention-based*, come il Transformer.

3.2.1 Attention is all you need

Il Transformer è un modello di deep learning completamente *attention-based* introdotto da Vaswani et al. nel 2017 nel paper "Attention is all you need"[11]. Il Transformer ha rivoluzionato il campo del deep learning per il linguaggio naturale, superando i modelli *encoder-decoder* in termini di performance e efficienza [9].

Il meccanismo di attenzione è il cuore del Transformer. Questo meccanismo permette al modello di assegnare pesi diversi alle parole nel testo in input, in modo da poter focalizzare l'attenzione sulle parole più rilevanti per il task in questione.

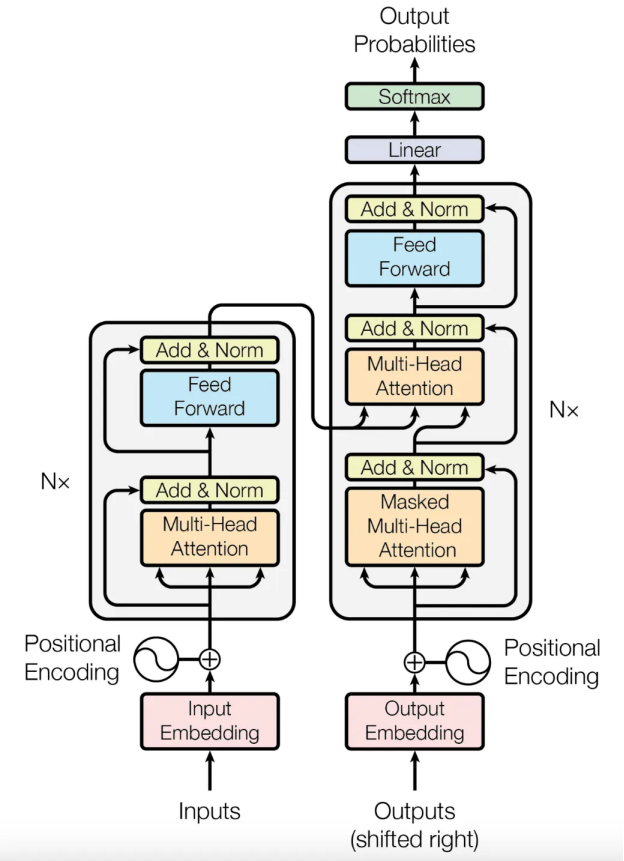

Figura 3.1: Architettura del Transformer

3.3 Embedding

Gli embedding sono una rappresentazione vettoriale di un oggetto in uno spazio vettoriale. Sono ampiamente utilizzati nel campo del deep learning per rappresentare parole, frasi, documenti e immagini in modo da poter essere usati come input per modelli di machine learning [4]. Gli embedding vengono creati utilizzando modelli di machine learning che imparano a mappare gli oggetti in uno spazio vettoriale in modo che oggetti simili siano vicini tra loro. Alcuni esempi di modelli di embedding sono Word2Vec, GloVe e FastText.

Alcuni utilizzi comuni degli embedding includono:

- **Elaborazione del linguaggio naturale**: embedding di parole e frasi per modelli di classificazione, clustering e generazione di testo. Consentono al modello di comprendere il significato delle parole.
- **Sistemi di raccomandazione**: utilizzare gli embedding per rappresentare utenti, prodotti, post in uno spazio vettoriale, in modo da poter calcolarne la similarità.
- **Ricerca di immagini**: permettono di comparare immagini tra di loro o di comparare immagini e testo. Questo permette di cercare immagini tramite testo o viceversa.
- **Ricerca semantica**: rendono possibile la ricerca di frasi o documenti simili in base al loro significato.

Eseguire operazioni su embedding è molto più efficiente se vengono memorizzati in un database apposito.

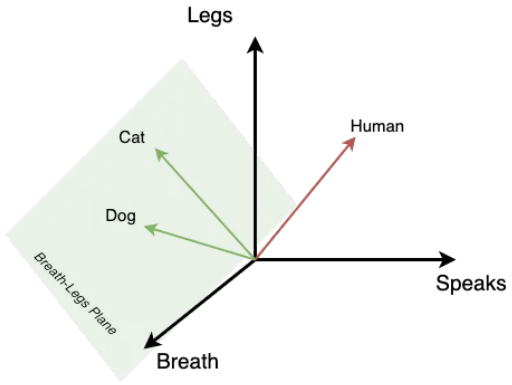

Figura 3.2: Esempio di embedding. Fonte: [1]

3.3.1 Vector Database

A differenza dei database tradizionali, i vector database sono progettati per memorizzare e interrogare dati in forma vettoriale. Questo permette di effettuare operazioni e manipolazioni su dati vettoriali in maniera più efficiente rispetto ad un database relazionale.

I vector database sono particolarmente utili per applicazioni che richiedono il calcolo della similarità tra oggetti in uno spazio vettoriale, come le applicazioni menzionate in precedenza.

Recentemente sono stati introdotti diversi vector database come **Pinecone**, **Chroma** e **MongoDB Atlas**, ma per questo lavoro è stato scelto **PostgreSQL** con il modulo *PGVector* per memorizzare gli embedding per semplicità di utilizzo e familiarità.

3.4 Fine-tuning di modelli

Il finetuning è una tecnica comune nel campo del deep learning per adattare un modello ad un task specifico.

Nel contesto del rilevamento di spoiler, il finetuning di un modello può essere utile per migliorare le performance del modello nel rilevare spoiler in un determinato contesto o linguaggio.

Il finetuning di un modello pre addestrato comporta l'addestramento del modello su un dataset specifico per un numero limitato di epoche. Questa tecnica consente di sfruttare le conoscenze già apprese dal modello durante l'addestramento iniziale, riducendo il tempo e le risorse necessarie per addestrare un nuovo modello da zero.

Durante il finetuning, i pesi del modello vengono aggiornati utilizzando un tasso di apprendimento più basso rispetto all'addestramento iniziale. Questo permette al modello di adattarsi al nuovo dataset senza dimenticare le conoscenze pregresse.

Ciò risulta particolarmente utile quando si dispone di un dataset limitato o quando si desidera adattare un modello pre addestrato a un dominio specifico.

3.5 Transizione all'implementazione

Avendo stabilito il contesto teorico e tecnologico nel capitolo precedente, il prossimo passo consiste nel descrivere come questi concetti sono stati applicati nella pratica. Nel prossimo capitolo ci focalizzeremo sugli aspetti pratici della realizzazione del sistema. Descriveremo in dettaglio il design e l'implementazione del sistema di rilevamento di spoiler, illustrando le sfide incontrate e le soluzioni adottate per creare un'applicazione efficace e utilizzabile.

Capitolo 4

Design e implementazione

4.1 Introduzione

Inizieremo con una panoramica generale dell'architettura del sistema, per poi passare a una descrizione dettagliata dei componenti principali.

4.2 Architettura del sistema

L'architettura del sistema è composta dalle seguenti componenti principali:

- **Frontend**: consiste in una estensione per il browser che permette di interagire con il sistema.
- **Backend**: è il server che gestisce le richieste provenienti dal frontend e comunica con il database e il LLM.
- **Embedder**: è il componente che genera gli embedding delle parole chiave che popolano il database, nonchè gli embedding delle richieste del frontend.
- **Database**: memorizza le informazioni relative agli embedding delle parole chiave che fungono da input al LLM.
- **LLM**: è il modello che genera le risposte in base alle richieste del frontend.

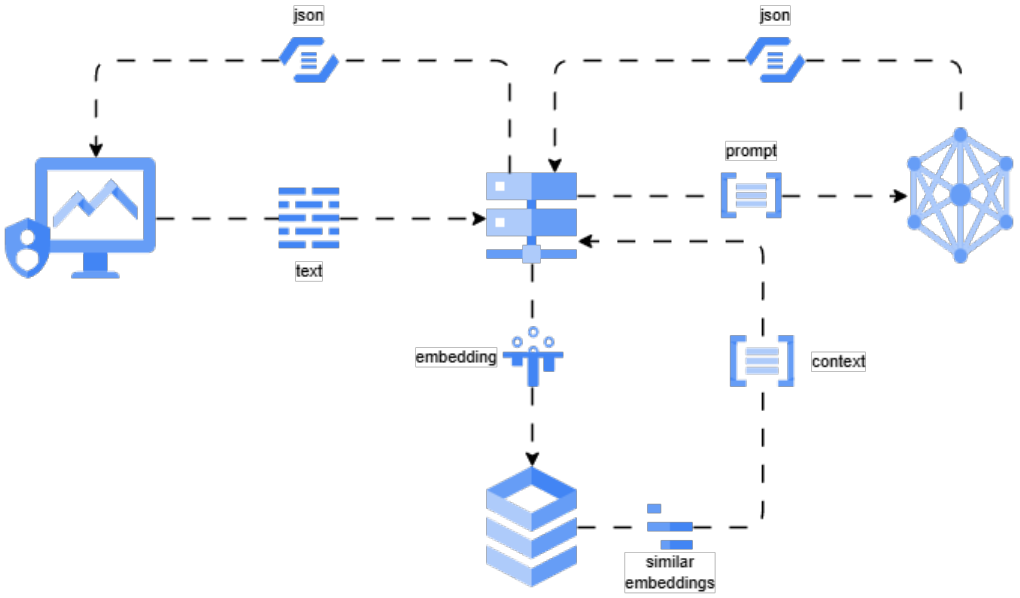

Figura 4.1: Architettura del sistema e flusso dei dati

4.3 Componenti principali

4.3.1 Frontend

Il frontend è un'estensione per il browser che consente di interagire con una qualsiasi pagina web. L'estensione è progettata per essere leggera e veloce, in modo da non influire sulle prestazioni del browser.

Il frontend è responsabile della raccolta delle informazioni, oltre che della visualizzazione dei risultati. Quando l'utente seleziona un elemento della pagina, il frontend estrae il contenuto testuale e lo invia al backend per l'elaborazione. Il backend genera quindi una risposta e la invia al frontend, che la visualizza all'utente.

Inoltre, il frontend permette di selezionare quale modello utilizzare per generare le risposte, in modo da poter testare le prestazioni di diversi modelli.

4.3.2 Backend

Il backend è il cuore del sistema, responsabile della gestione delle richieste provenienti dal frontend e della comunicazione con il database e il LLM. Il backend è implementato in Python e utilizza il framework FastAPI che permette la creazione di API RESTful, per applicazioni web scalabili e perfor-

manti. Il backend è progettato per essere modulare e facilmente estensibile, in modo da poter aggiungere nuove funzionalità in futuro.

Quando il backend riceve una richiesta dal frontend, esegue le seguenti operazioni:

- Preprocessa il contenuto testuale dalla richiesta
- Tokenizza il testo dividendo il testo in frasi più piccole.
  Questo passaggio è fondamentale per creare il contesto necessario per il LLM.
- Genera gli embedding di ogni frase attraverso l'embedder.
- Esegue una ricerca nel database per trovare le parole chiave più simili agli embedding generati
- Invia la richiesta al LLM per generare la risposta. La richiesta al LLM include il contenuto processato e le parole chiave trovate nel database.
- Restituisce la risposta del LLM al frontend sotto forma di JSON.

La comunicazione tra il server e il LLM avviene tramite la libreria di OpenAI, che consente di inviare richieste al modello e ricevere le risposte utilizzando una interfaccia configurabile. In particolare, la libreria permette di utilizzare qualiasi modello compatibile con l'interfaccia di OpenAI e questo ha permesso di testare diversi modelli attraverso il tool **Ollama**.

4.3.3 Ollama

Ollama è un tool open source che consente di eseguire LLM localmente, senza la necessità di una connessione a Internet. Ollama stato è progettato per essere semplice da usare e consente di eseguire modelli LLM in modo rapido e efficiente sfruttando l'accelerazione hardware delle GPU quando possibile. Ollama offre una varieta di modelli LLM pre-addestrati, tra cui `Llama`, `Mistral`, `Phi`, `Deepseek` e `Gemma`.

In particolare, `Gemma` è il modello utilizzato per testare il sistema, in quanto è ottimizzato per l'uso in locale e offre prestazioni eccellenti rispetto a modelli di dimensioni comparabili.

4.3.4 Embedder

L'embedder è il componente responsabile della generazione degli embedding delle parole chiave che popolano il database, nonchè degli embedding delle

richieste del frontend. L'embedder è implementato in Python e utilizza la libreria `SentenceTransformers`[8] per generare gli embedding delle parole chiave e delle richieste.

Attraverso l'uso di `SentenceTransformers` è stato possibile eseguire il fine-tuning del modello di embedding `all-MiniLM-L6-v2` per ottimizzare le prestazioni del sistema. Il fine-tuning del modello è stato eseguito utilizzando un dataset autonomamente creato, tramite l'uso di un web scraper che ha estratto le informazioni da fonti pubbliche disponibili su Internet.

Il dataset è composto da una serie di articoli e documenti che sono stati manipolati per estrarre il contenuto testuale e le parole chiave.

4.4 Dataset e tool utilizzati

Qui di seguito sono riportati i tool e le librerie utilizzati per la creazione del dataset e il fine-tuning del modello di embedding:

- **Puppeteer**: è una libreria Node.js che consente di controllare un browser Chrome o Chromium in modo programmatico.
- **Pandas**: è una libreria Python per la manipolazione e l'analisi dei dati.
- **Visualizer**: un tool creato appositamente per visualizzare i risultati dello scraping, utilizzato per verificare la correttezza dei dati estratti e studiare le relazioni tra le parole chiave e i documenti.

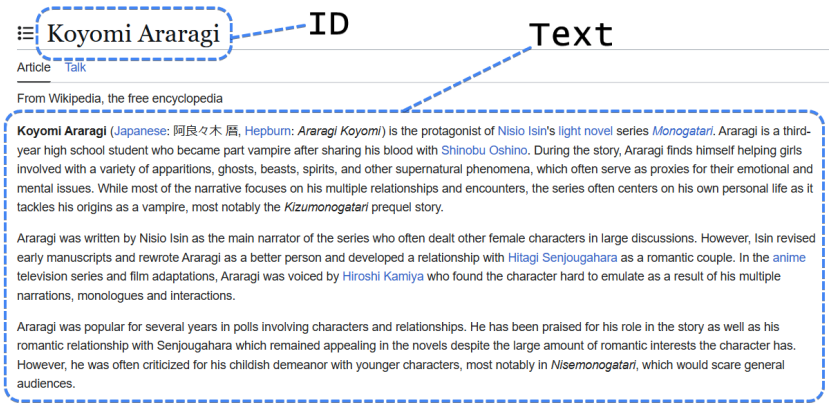

Figura 4.2: fonte: Wikipedia

In figura 4.2 è mostrato un esempio di scraping effettuato attraverso il tool, che ha estratto il contenuto testuale di una pagina Wikipedia.

Notiamo che il tool è in grado di estrarre non solo il contenuto testuale, ma anche i collegamenti ad altre pagine presenti nei singoli paragrafi, unitamente a una serie di metadati. Ciò ha reso possibile la creazione di un grafo delle relazioni tra le parole chiave e i documenti, che è stato utilizzato per il fine-tuning del modello di embedding. Il grafo è stato creato utilizzando la libreria `D3.js`, che consente di creare visualizzazioni interattive dei dati.

4.4.1 Visualizer

Il visualizer è stato creato appositamente per studiare le relazioni tra le parole chiave e i documenti, in modo da ottimizzare il fine-tuning del modello di embedding creando un dataset bilanciato e di qualità superiore.

Figura 4.3: Esempio di grafo creato dal visualizer

In figura 4.3 è mostrato un esempio di grafo. Ogni nodo rappresenta un documento e ogni arco rappresenta una relazione tra due documenti.

Possiamo osservare che alcuni documenti sono più connessi di altri, il che indica che hanno diversi collegamenti, o che diversi documenti fanno riferimento a loro. E' anche possibile osservare che alcuni documenti sono isolati, il che indica che non hanno collegamenti con altri documenti.

Attraverso il visualizer è stato possibile filtrare i documenti in base a diversi criteri, come ad esempio il numero di collegamenti, metadati, o altri tipi di filtro.

Nella scelta dei documenti da includere nel dataset si è tenuto conto di diversi fattori statistici [7] [5] [6], elencati di seguito:

Statistica	Valore
Coefficiente di assortatività	-0.4677
Mescolanza di assortatività discreta	-0.0059
Diametro del grafo	9
Densità del grafo	0.024
Lunghezza media del percorso	3.85
Nodi isolati	76

Tabella 4.1: Statistiche globali del grafo

Statistica	Valore
Coefficiente di Clustering	0.333
Componenti nel Cluster	258
Centralità di Closeness	0.365

Tabella 4.2: Statistiche per singoli nodi

Nota: I valori riportati nelle tabelle 4.1 e 4.2 sono stati generati su un dataset di esempio per illustrare le funzionalità del visualizer.

4.5 Preparazione del dataset

Per preparare il dataset è stato necessario eseguire diverse operazioni di post-processing sui dati estratti dallo scraping.

In particolare, sono stati eseguiti i seguenti passaggi:

- Rimozione dei metadati
- Normalizzazione del testo
- Oversampling di alcune parole chiave
- Introduzione di rumore nel dataset
- altre operazioni (ulteriori dettagli sono disponibili nel codice sorgente)

Si è deciso di eseguire l'oversampling di alcuni tipi di parole chiave, in modo da bilanciare il dataset e ottenere un modello di embedding più robusto.

Un esempio di oversampling è mostrato in figura 4.4.

Figura 4.4: Esempio di oversampling

Il risultato finale del dataset è stato un file JSON con il seguente formato:

anchor	positive	negative
koyomi araragi	vampiro, studente, adolescente	nobile
shinobu oshino	vampiro, 500 anni, nobile	studente
ononoki	sissi, yay. peace peace! 👆👁_👁👆	hatsune miku
bakemonogatari	anime, manga, light novel	film
nise	anime, manga, light novel	film

Tabella 4.3: Formato del dataset

Nota: I dati riportati nella tabella 4.3 sono stati generati su un dataset di esempio per illustrare il formato del dataset.

In particolare, la colonna `anchor` contiene la parola chiave, la colonna `positive` contiene il contesto che si desidera associare all'ancora, mentre la colonna `negative` contiene l'elemento che si desidera evitare di associare alla parola chiave.

Tramite l'uso di questo formato è stato possibile separare efficacemente le parole chiave e i contesti, in modo da ottenere un dataset qualitativamente migliore.

4.5.1 Scelta dei negativi

La scelta degli esempi negativi è stata cruciale per il fine-tuning del modello di embedding.

Esistono principalmente due approcci per la scelta degli esempi negativi:

- **Scelta casuale**: consiste nel selezionare casualmente gli esempi negativi dal dataset.
- **Scelta non casuale**: consiste nel selezionare gli esempi negativi in base a regole stabilite.

Nel primo approccio, la scelta degli esempi negativi è completamente casuale e non tiene conto delle relazioni tra le parole chiave e i contesti. Questo approccio può portare a risultati non ottimali, in quanto gli esempi negativi potrebbero non essere rappresentativi del contesto in cui si desidera utilizzare il modello di embedding.

La strategia adottata nel progetto è stata quella di estrarre randomicamente il contesto a partire dalle parole chiave "vicine" all'ancora di riferimento. Questo approccio tiene conto delle relazioni tra i diversi documenti in modo da ottenere degli embedding più robusti e rappresentativi.

Nel nostro caso, a partire da un nodo del grafo, l'ancora è rappresentata dal titolo del nodo, l'esempio positivo è rappresentato dal contenuto del nodo, mentre l'esempio negativo è rappresentato dal contenuto di un nodo adiacente.

Questo approccio ha portato a risultati migliori rispetto all'approccio casuale poichè sfrutta il fatto che i documenti che hanno collegamenti in comune, molto probabilmente condividono informazioni simili. Permettendo al modello di distinguere tra i due, è stato possibile ottenere un embedding più robusto e rappresentativo.

4.6 Fine-tuning del modello

Il fine-tuning del modello di embedding è stato eseguito utilizzando la libreria `SentenceTransformers` e il dataset creato in precedenza. Il fine-tuning è stato eseguito utilizzando il modello `all-MiniLM-L6-v2`, che è un modello di embedding pre-addestrato basato su `BERT` e ottimizzato per l'uso in locale. Il fine-tuning è stato eseguito utilizzando il dataset creato in precedenza e il modello è stato addestrato per un numero di epoche pari a 4.

La funzione di penalità utilizzata è stata la `TripletLoss`[2], comunemente utilizzata per l'addestramento di modelli di embedding. La `TripletLoss` è progettata per minimizzare la distanza tra l'ancora e l'esempio positivo, e massimizzare la distanza tra l'ancora e l'esempio negativo.

Come discuteremo nel prossimo capitolo, il fine-tuning del modello ha permesso di ottenere un modello di embedding robusto e rappresentativo.

Capitolo 5

Valutazione

5.1 Introduzione

In questo capitolo, presentiamo i risultati ottenuti durante la fase di valutazione del nostro sistema.

Inizieremo con una panoramica dei dati utilizzati per il fine-tuning e la valutazione del modello, seguita da una descrizione delle metriche di valutazione impiegate.

Successivamente, presenteremo i risultati ottenuti, analizzando le prestazioni del nostro sistema in termini di accuratezza e capacità di rilevamento degli spoiler.

Discuteremo le implicazioni dei risultati e le possibili direzioni future per il miglioramento del sistema.

5.1.1 Specifiche tecniche

La configurazione del sistema è stata eseguita su un computer con le seguenti specifiche:

- CPU: AMD Ryzen 7 5800H (8 core, 16 thread)
- GPU: NVIDIA GeForce RTX 3070 Laptop (8 GB)
- RAM: 16 GB
- Sistema operativo: Windows 10 22H2
- Versione di Python: 3.12.9

5.2 Dataset e metodologia

Per la valutazione del nostro sistema, abbiamo utilizzato 2 strategie: **pair** e **triplet**. Il dataset **pair** è derivato da **triplet** escludendo la colonna *negative*. Questo ha permesso di valutare il sistema in diverse configurazioni.

Il codice per la generazione degli split e il training è identico per entrambe le strategie, con l'unica differenza che per **pair** il dataset è stato ridotto a 2 colonne: *positive* e *negative*, adattando la funzione di loss di conseguenza. Il seed utilizzato per la generazione degli split è 298[3], garantendo la riproducibilità dei risultati.

Parametro	**Valore**
Dataset	4067 righe
Split	80% train, 20% test
Epoche	4
Loss (pair)	CosineSimilarityLoss
Loss (triplet)	TripletLoss

Tabella 5.1: Panoramica dei parametri utilizzati per il fine-tuning e la valutazione del modello.

5.3 Embedders

Durante la creazione del sistema, si è deciso di utilizzare due modelli pre-addestrati per generare gli embedding:

- **all-MiniLM-L6-v2**
- **all-mpnet-base-v2**

Nessuno dei modelli è stato addestrato su dati specifici al dominio degli spoiler o sui dati utilizzati per il fine-tuning. Come mostrano i risultati di seguito, nessuno dei modelli soddisfaceva i requisiti per il task dato il dominio specifico, ma con un fine-tuning adeguato, sono stati in grado di generare embeddings soddisfacenti.

Entrambi i modelli sono disponibili su HuggingFace, installando la libreria `sentence-transformers`. I modelli scelti sono comunemente utilizzati per la creazione di embeddings per frasi o paragrafi brevi, rendendoli adatti al nostro task. Il codice per il fine-tuning e la valutazione è stato scritto in Python ed è disponibile su GitHub.

5.3.1 Risultati

Nota: I migliori risultati sono segnalati con un asterisco (*)

Pair

Modello	Similarità positiva
Pre-addestramento	
all-mpnet-base-v2	0.1554
all-MiniLM-L6-v2*	**0.1688**
Fine-tuned	
all-mpnet-base-v2*	**0.9975**
all-MiniLM-L6-v2	0.9938

Tabella 5.2: Risultati della valutazione dei modelli di embedding (Pair)

Triplet

Modello	Similarità positiva	Similarità negativa
Pre-addestramento		
all-mpnet-base-v2	0.1554	0.2001
all-MiniLM-L6-v2*	**0.1688**	**0.2113**
Fine-tuned		
all-mpnet-base-v2*	**0.6358**	**-0.8118**
all-MiniLM-L6-v2	0.6274	-0.7221

Tabella 5.3: Risultati della valutazione dei modelli di embedding (Triplet)

Notiamo che i risultati pre-addestramento sono molto bassi, e simili tra loro, il che indica che i modelli non sanno distinguere categorizzare bene i dati in input. Tuttavia, dopo il fine-tuning, i risultati sono significativamente migliorati, con all-mpnet-base-v2 che ottiene i migliori risultati in entrambe le configurazioni. Da notare che i risultati nella configurazione **pair** sono migliori rispetto a quelli nella configurazione **triplet**, ma ciò indica solo che i modelli hanno appreso a generare embedding più simili tra loro, **non necessariamente più accurati**.

5.3.2 Considerazioni

In definitiva, i risultati mostrano che il fine-tuning ha portato a un miglioramento significativo delle prestazioni, specialmente nella configurazione **triplet**, dove i modelli sono stati in grado di generare embedding con una similarità positiva superiore a 0.64 e una similarità negativa inferiore a -0.81. Ciò indica che i modelli sono stati in grado di raggruppare correttamente i dati simili e separare quelli dissimili, dimostrando la loro efficacia in questo task dopo un adeguato fine-tuning.

Nononstante all-mpnet-base-v2 abbia ottenuto, di poco, risultati migliori rispetto a all-MiniLM-L6-v2, quest'ultimo ha dimostrato di essere più veloce e più leggero, rendendolo la scelta preferita per questo progetto. Inoltre, all-MiniLM-L6-v2 ha mostrato prestazioni migliori in fase di fine-tuning.

5.3.3 Prestazioni in fase di fine-tuning

Modello	Tempo (min:sec)	Step	Perdita
all-mpnet-base-v2	03:26	500	3.8035
all-MiniLM-L6-v2	00:53	500	4.0104

Tabella 5.4: Tempi e perdite del fine-tuning (Triplets)

Modello	Tempo (min:sec)	Step	Perdita
all-mpnet-base-v2	02:10	500	0.0183
all-MiniLM-L6-v2	00:37	500	0.0263

Tabella 5.5: Tempi e perdite del fine-tuning (Pairs)

Le prestazioni di all-MiniLM-L6-v2 sono decisamente migliori rispetto a quelle di all-mpnet-base-v2 in termini di tempo. Il tempo di inferenza è un aspetto critico per l'implementazione di modelli che devono essere utilizzati in locale, e questo modello ha dimostrato di essere la scelta più adatta per il nostro scopo.

5.4 LLM

Per la valutazione del modello LLM, è stato scelto **Gemma3**.

5.4.1 Gemma3

Gemma3 è un modello LLM open-weight sviluppato da Google, progettato per essere altamente efficiente e performante. Gemma3 ha capacità multimodali, il che significa che può elaborare testo e immagini.

Data la sua architettura avanzata derivante da Gemini, lo stato dell'arte dei modelli di Google, Gemma3 è in grado di comprendere e generare contenuti con performance competitive rispetto a modelli di dimensioni molto maggiori[10].

Per questo progetto, non abbiamo utilizzato la modalità multimodale, ma ci siamo concentrati sulla generazione di testo.

Si è deciso di utilizzare Gemma3 per la sua capacità di generare testo di alta qualità e la sua architettura leggera che consente un'implementazione rapida ed efficiente a latenza ridotta.

A causa di limitazioni hardware, non è stato possibile scaricare il modello completo, quindi è stato scelto di utilizzare versioni del modello con dimensioni diverse. Gemma3 è disponibile in diverse dimensioni:

- **Gemma3-1B**: 1 miliardo di parametri
- **Gemma3-4B**: 4 miliardi di parametri
- **Gemma3-12B**: 12 miliardi di parametri
- **Gemma3-27B**: 27 miliardi di parametri

La versione 1B non possiede la capacità multimodali, mentre la versione 12B ha una dimensione $> 8GB$, per cui non è stato possibile utilizzare appieno il modello sfruttando l'accelerazione GPU. Per questo motivo, sono stati utilizzati principalmente i modelli 1B e 4B.

5.4.2 Risultati

Per valutare le prestazioni di Gemma3, è stato utilizzato un dataset diverso rispetto a quello utilizzato per il fine-tuning e la valutazione del modello di embedding.

Capitolo 6

Conclusioni

Bibliografia

[1] Oleg Borisov. Word embeddings: Intuition behind the vector representation of the words. https://medium.com/data-science/word-embeddings-intuition-behind-the-vector-representation-of-the-words-7e4eb2410bba/, 2020.

[2] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification, 2017.

[3] Hachikuji Mayoi. Nisemonogatari, karen bee, part one, Jan 2012. Trasmissione televisiva.

[4] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 3781, 2013.

[5] M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89, 2002.

[6] M. E. J. Newman. Mixing patterns in networks. *Physical Review E*, 67, 2003.

[7] M. E. J. Newman. *Networks: an introduction*. Oxford University Press, 2010.

[8] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[9] Google Research. Transformer: A novel neural network architecture for language understanding. https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/, 2017.

[10] Gemma Team. Gemma 3, 2025.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.