PROGETTO NLP | 2023

TED TALKS PER L'ITALIANO

Alessandro Quirile N97/402
Ivano Matrisciano N97/392

Università degli Studi di Napoli Federico II

https://github.com/alessandroquirile/Ted-Talks

1. Introduzione

Il nostro obiettivo è quello di realizzare un sistema per la serializzazione e l'analisi semantica del dataset TED Talks per l'italiano. Il sistema è in grado di trattare sia dati testuali che dati vocali ed utilizza un DBMS vettoriale open-source fault tolerant, Weaviate, per memorizzarli sotto forma di embedding e consentirne agilmente il retrieval.

Il sistema permette l'interrogazione del database, fornendo in output anche un riassunto degli elementi trovati, generato a partire dalle trascrizioni delle conferenze.

Le ricerche che il sistema può eseguire sono le seguenti:

- Ricerca vettoriale e semantica
- Ricerca generativa (Question & Answering)
- Ricerca ibrida
- Ricerca tramite sample audio

A tal proposito Weaviate mette a disposizione varie metriche tra cui:

- Similarità coseno: quella di default. Il range è [0;2]. 0 indica vettori identici mentre 2 indica vettori completamente opposti
- Similarità basata su prodotto scalare cambiato di segno: il range è (-inf,+inf). Ad esempio -3 è considerato uno score di similarità migliore di -2 e +2 è migliore di +5
- Distanza euclidea: il range è [0; +inf). 0 indica vettori identici
- Distanza di Hamming: il range è [0; +inf). 0 indica vettori identici
- Distanza di Manhattan: il range è [0; n_dims). 0 indica vettori identici

In generale misure più alte indicano minore similarità. Viceversa, misure più basse indicano maggiore similarità.

Abbiamo scelto la metrica di similarità basata su coseno in quanto, a runtime, tutti gli embedding coinvolti nella ricerca vengono trasformati in versori (vettori di lunghezza unitaria) i quali sono particolarmente efficienti da gestire. Inoltre la similarità basata su coseno risulta ottimizzata per i sistemi Linux, Darwin con il set di istruzioni AVX2. Infine è stata scelta perché il range è molto più ristretto di quello delle altre opzioni; se da un punto di vista teorico-matematico questo potrebbe non essere rilevante, da un punto di vista implementativo potrebbe evitare problemi di under/overflow.

È possibile cambiare la metrica di similarità modificando il campo `vectorIndexConfig` a valle della creazione di uno schema. Si può addirittura pensare di creare una nuova metrica *ad-hoc* come combinazione di alcune già esistenti.

Infine, sebbene non esplicitamente richiesto nella traccia, il nostro DBMS mette a disposizione anche una funzionalità per sintetizzare testo non eccessivamente lungo. In caso contrario Weaviate potrebbe non rispondere per via dei suoi limiti tecnologici; ad ogni modo è possibile bypassare questo problema dividendo a metà il testo e richiedendo la sintesi della prima metà e la sintesi della seconda metà (ed eventualmente la sintesi delle due sintesi).

2. Weaviate

Questo capitolo fornisce alcune considerazioni di base su Weaviate per capirne le funzionalità e usarle coerentemente durante lo sviluppo del sistema.

2.1 Come memorizzare i dati?

I dati (data object) vengono associati a delle 'classi' che possono avere una o più proprietà e sono serializzati nel formato JSON:

```
{
    "name": "Alice Munro",
    "age": 91,
    "born": "1931-07-10",
    "wonNobelPrize": true,
    "description": "She is a famous story-writer"
}
```

Possiamo specificare manualmente un embedding per ciascun data object:

```
{
    "id": "779c8970",
    "class": "Author",
    "properties": {
        "name": "Alice Munro",
        (...)
    },
    "vector": [
        -0.16,
        -0.06
    ]
}
```

Oppure possiamo utilizzare il modulo *vectorizer* per far sì che sia il dbms stesso a calcolare gli embedding degli oggetti non appena questi vengono inseriti nella base dati. Weaviate mette a disposizione diversi moduli, tra questi è stato scelto *text2vec-transformers* perché è quello che permette di utilizzare modelli self-hosted. Parleremo dei vectorizer nei paragrafi successivi.

Una volta che abbiamo definito un autore, possiamo memorizzare una collezione di più data object attraverso una sintassi *array-like*:

```
[ {...}, {...}, ..., {...}]
```

Dove ciascun `{...}` rappresenta un data object. Ciascun oggetto memorizzato in Weaviate ha un UUID che lo identifica universalmente.

2.2 Come memorizza le relazioni?

Può capitare che un data object abbia delle relazioni con un altro data object. Weaviate serializza questa informazione attraverso la *cross-reference: a*d esempio se vogliamo serializzare l'informazione che "Paul Krugman scrive per il New York Times" possiamo farlo memorizzando dapprima un oggetto *Publication* e poi specificare la relazione per l'altro. Quindi:

```
{
    "id": "32d5a368",
    "class": "Publication",
    "properties": {
        "name": "The New York Times"
    },
    "vector": [...]
```

```
}
```

e poi

```
{
    "id": "779c8970",
    "class": "Author",
    "properties": {
        "name": "Paul Krugman",
        ...
        "writesFor": [
            {
                "beacon": "weaviate://localhost/32d5a368",
                "href": "/v1/objects/32d5a368"
            }
        ],
    },
    "vector": [...]
}
```

notando che abbiamo utilizzato lo UUID come chiave.

Prima di aggiungere un data object occorre definire un *Weaviate schema* che formalizza la sua struttura. Nel caso in cui si stia aggiungendo un data object il cui schema non è stato ancora definito, Weaviate è in grado di inferirlo in automatico.

2.3 Come calcola gli embedding?

Abbiamo visto che l'embedding è in generale opzionale oppure può essere caricato manualmente. In realtà Weaviate mette a disposizione dei moduli, i *vectorizer*, che specificano in che modo un embedding debba essere calcolato (automaticamente).

A seconda del tipo di dato (testo, audio, immagini...) e del caso d'uso (question answering, ricerca per similarità...) possiamo scegliere un vectorizer che meglio si adatta alle nostre esigenze.

I moduli vectorizer (anche chiamati *dense retriever*) trasformano, come detto, i dati nella loro rappresentazione vettoriale. Tra i vectorizer per testi, Weaviate mette a disposizione diversi moduli per chiamare le API di servizi terzi quali OpenAI o HuggingFace. Il modulo vectorizer *text2vec-transformers* permette di accedere ad un vettorizzatore self-hosted, tramite un'architettura a microservizi basata su Docker o Kubernetes. Tramite questo modulo è possibile sfruttare diversi modelli di NLP pre-addestrati come BERT, DistilBERT, RoBERTa e DistilRoBERTa. Ciascun modello viene incapsulato in un container Docker per questioni di efficienza, scalabilità e sicurezza (ciascun servizio viene eseguito nel contesto di una sandbox). Per scegliere il modello che fa al caso nostro occorre scegliere il container Docker corretto. Alcuni container sono già disponibili; altri, invece, possono essere istanziati con un file di configurazione chiamato *Dockerfile*.

I transfomer vengono eseguiti *molto* più velocemente con le GPU rispetto alle CPU. Senza l'uso delle GPU, infatti, ricerche di similarità basate su `near_text` potrebbero risultare un collo di bottiglia.

Abbiamo scelto di utilizzare il modello *sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2* perché risulta un ottimo compresso tra prestazioni e velocità; inoltre è multilingua.

3. Scelta dei modelli ed estrazione delle feature

In questo capitolo verranno descritti i modelli di machine learning utilizzati per i vari moduli che compongono il sistema.

3.1 Modello text2vec

Come vettorizzatore per il testo è stato scelto un modello multilingua basato su Sentence BERT (SBERT) noto con il tag *sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2* in quanto ufficialmente supportato da Weaviate ed offre un ottimo compromesso tra performance e tempi d'esecuzione, producendo embedding contenuti (solo 384 elementi), permettendo così di risparmiare spazio su disco. I modelli Sentence-BERT offrono un miglioramento notevole delle performance rispetto ai BERT classici, portando in alcuni casi il tempo di esecuzione per una similarità da 65 ore a 5 secondi, mantenendo la stessa accuratezza [M1].

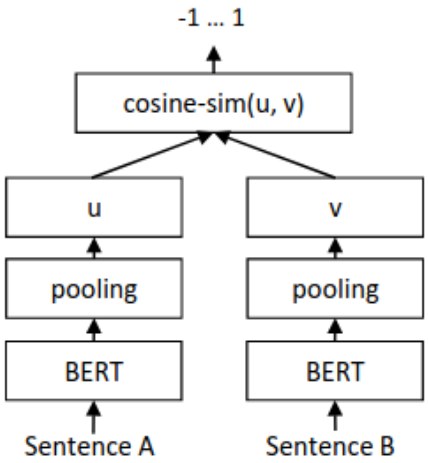


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

I sentence transformer permettono di mappare un testo in un vettore di embedding che ne racchiude le informazioni semantiche. Così facendo è possibile confrontare tali vettori per ottenere la similarità tra due testi.

3.2 Modello di Question Answering

Lavorando con testi in italiano è imperativo che il modello di question answering supporti la lingua italiana. Purtroppo tutti i modelli messi a disposizione ufficialmente da Weaviate lavorano unicamente con l'inglese. Per questo motivo si è reso necessario creare un container che implementi un modulo di Q&A usando un modello scelto ad hoc.

Come modello di question & answering è stato scelto [timpal0l/mdeberta-v3-base-squad2](). mDeBERTa è una versione multilingua di DeBERTa, lavoro che migliora RoBERTa incorporando degli strati di disentangled attention. Il training è effettuato cercando di ricostruire token corrotti in input. Questo meccanismo, tipicamente implementato con un transformer in reti come BERT, è stato sostituito con un modulo di Replaced Token Detection (RTD) ispirato ad ELECTRA [M4]. Questo modulo prevede di usare un approccio simile alle Generative Adversarial Network per corrompere i token da un lato ed individuare i token corrotti dall'altro. [M3]

Il modulo di Question & Answer da integrare in Weaviate è stato generato lanciando lo script `qna-module-build.sh`, contenente i seguenti comandi:

```
git clone
https://huggingface.co/timpal0l/mdeberta-v3-base-squad2
docker build -f qna-module.Dockerfile -t qna-module .
```

Lo script esegue prima il download del modello da huggingface, dopodiché avvia la costruzione di un container come indicato in `qna-module.Dockerfile` qui riportato:

```
FROM semitechnologies/qna-transformers:custom
```

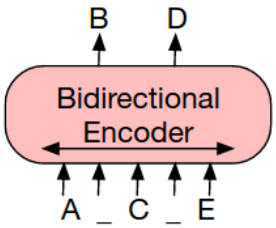
```
COPY ./mdeberta-v3-base-squad2 /app/models/model
```

Il dockerfile sfrutta l'immagine fornita da weaviate per la costruzione di moduli custom e ne copia il modello nella cartella `/app/models/model`

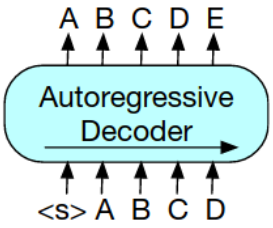
3.3 Modello di Summarization

Come summarizer è stato scelto [facebook/bart-large-cnn](), un modello BART messo a punto sul quotidiano Daily Mail. Il modello accoppia un encoder basato su BERT e un decoder basato su GPT. L'encoder ha lo scopo di condensare le informazioni semantiche presenti nel testo in un embedding di dimensione strettamente più piccola rispetto all'input. Tale embedding verrà poi fornito al decoder con lo scopo di ricostruire il testo. Questa struttura ha possibilità generative.

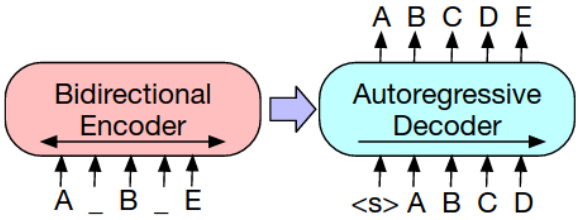
Di seguito viene riportata la figura 1 del paper [M2] che illustra l'architettura concettuale del modello BART.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 1: A schematic comparison of BART with BERT (Devlin et al., 2019) and GPT (Radford et al., 2018).

L'architettura di BART segue il meccanismo delle attenzioni così come descritto nel paper *Attention Is All You Need* [5] , usando però *GeLU* come

funzione di attivazione. In più il decoder include anche un layer di cross-attention, ma, a differenza di BERT, non include un layer di feed-forward [M2]

Questo modello permette di gestire testi con un numero di token non superiore a 1024. Volendo perciò riassumere le trascrizioni delle conferenze, è necessario spezzare il testo in più parti e riassumere ogni parte individualmente. Per questo motivo il task di summarization è stato spostato a livello applicativo, usando però lo stesso modello qui descritto.

3.4 Estrazione delle feature audio

Weaviate non permette di lavorare direttamente con sample audio. È stato perciò necessario calcolare gli embedding a livello applicativo e storicizzare gli oggetti fornendo manualmente il vettore al DBMS.

Per l'estrazione delle feature audio si era inizialmente pensato di usare un modello come Speech2Vec [M5] che promette di raccogliere informazioni semantiche simili a quelle estratte da word2vec, ma lavorando direttamente con l'audio, senza perciò passare per la trascrizione del testo stesso. A causa però della scarsità di informazioni riguardo questo modello e ad alcune controversie [CON] inerenti la sua efficacia, è stato scelto un approccio di tipo differente.

Le feature audio sono state estratte mediante l'utilizzo di un modello audio basato su wav2vec2. Inizialmente si era pensato al modello facebook/wav2vec2-large-xlsr-53, modello multilingua addestrato su common_voice. Successivamente si è passati alla sua versione ridotta nota come facebook/wav2vec2-base-100k-voxpopuli per ridurre i tempi di calcolo. Quest'ultimo modello, anch'esso multilingua, è stato addestrato su 100 ore di audio proveniente dal dataset voxpopuli [VOX].

Le feature di un sample audio vengono calcolate facendo girare il modello audio direttamente sul sample. Questo ci restituisce in output un vettore di feature da 512 elementi per ogni finestra da 20ms del sample audio. Similmente a quanto fatto a lezione con word2vec, si calcola l'embedding dell'intero sample come la media degli embedding degli spezzoni che lo compongono.

```
extractor_data = self.feature_extractor(audio_chunk,
sampling_rate=self.feature_extractor.sampling_rate, padding=True,
return_tensors="pt")

model_output =
self.audio_model(extractor_data.input_values.to(self.device))
chunk_features = torch.mean(model_output.extract_features, axis=1)
```

Per limitare l'utilizzo della memoria, i file audio vengono caricati e processati in spezzoni (chunk) da 60 secondi l'uno in base alla seguente procedura:

1. Per ogni spezzone di audio:
    a. Carica la forma d'onda ed esegui un resample a 16kHz
    b. Trasforma la forma d'onda in un tensore pytorch
    c. Usa il modello d'audio per generare le feature per ogni finestra da 20ms
    d. Fai la media di queste feature
2. Restituisci come feature del file audio la media delle feature degli spezzoni che compongono il file stesso

L'obiettivo, in questo caso, è stato trovare un compromesso tra complessità in termini di risorse (come memoria e tempi di elaborazione) e prestazioni.

```
def extract_long_audio_embedding(self, file_path) -> np.array:
   original_sample_rate = librosa.get_samplerate(file_path)
   resample_sample_rate = self.feature_extractor.sampling_rate

   chunk_embeddings = []  # Contiene le feature di ogni chunk audio
(60 secondi)
   splitted_audio_chunks = self._get_audio_streams(file_path,
original_sample_rate, chunk_size_seconds=60)
   for audio_chunk in splitted_audio_chunks:
       # Ricampiona il chunk e lo converte in un tensore pytorch
       audio_chunk = librosa.resample(audio_chunk,
orig_sr=original_sample_rate, target_sr=resample_sample_rate)
       audio_chunk = torch.tensor(audio_chunk).to(self.device)

       # Estrae le feature da dare al modello
       extractor_data = self.feature_extractor(audio_chunk,
sampling_rate=self.feature_extractor.sampling_rate, padding=True,
return_tensors="pt")

       with torch.no_grad():
           try:
               # Ottiene le feature audio model
               model_output =
self.audio_model(extractor_data.input_values.to(self.device))

               chunk_features =
torch.mean(model_output.extract_features, axis=1)
               chunk_features = np.array(chunk_features.cpu())  #
Converte in array numpy
               chunk_embeddings.append(chunk_features)
           except:
               print("Errore nel processing del chunk")

   # Combina le feature di ciascun chunk facendo la media degli
embedding
   file_embedding = np.mean(chunk_embeddings, axis=0)
   return file_embedding
```

4. Inizializzazione del sistema

4.1 Installazione dei requisiti

Raggiungi https://github.com/alessandroquirile/Ted-Talks e clona la repository. In seguito dovrai installare Docker per avviare un'istanza di Weaviate e poi dovrai scaricare le dipendenze del progetto.

Per installare Docker puoi accedere al sito ufficiale oppure, se hai Linux, puoi lanciare il seguente comando dal terminale:

```
$ sudo apt install docker docker-compose
```

Per il download del modello da huggingface:

```
$ sudo apt install git-lfs
```

Infine per installare le dipendenze del progetto puoi usare il gestore di pacchetti python `pip`:

```
$ pip install -r requirements.txt
```

4.2 Configurazione di weaviate

Weaviate è stato configurato attivando i seguenti moduli:

- **text2vec-transformers:** modulo principale per la vettorizzazione dei dati, come già discusso
- **qna-transformers**: modulo per question & answer, quando abilitato permette di interrogare la base di dati per ottenere risposte dirette alle domande fornite. Ad esempio: "Di quale film Ravensburger ha creato un gioco da tavolo?"
- **ref2vec-centroid**: permette di vettorizzare una classe facendo una media dei vettori delle classi "cross-referenziate"

Un ulteriore modulo **sum-transformers** è stato spostato a livello applicativo per motivi che verranno descritti in seguito. Questo modulo di summarization permette di creare dei riassunti partendo dalle informazioni contenute nella base dati.

La configurazione descritta è stata implementata con il seguente contenuto per il file `docker-compose.yml`

```
---
version: '3.4'
services:
  weaviate:
      command:
      - --host
      - 0.0.0.0
      - --port
      - '8080'
      - --scheme
      - http
      image: semitechnologies/weaviate:1.19.8
      ports:
      - 8080:8080
      restart: on-failure:0
      environment:
      TRANSFORMERS_INFERENCE_API: 'http://t2v-transformers:8080'
      QNA_INFERENCE_API: 'http://qna-transformers:8080'
      SUM_INFERENCE_API: 'http://sum-transformers:8080'
      QUERY_DEFAULTS_LIMIT: 25
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true'
      PERSISTENCE_DATA_PATH: '/var/lib/weaviate'
      DEFAULT_VECTORIZER_MODULE: 'text2vec-transformers'
      ENABLE_MODULES:
'text2vec-transformers,qna-transformers,ref2vec-centroid'
      CLUSTER_HOSTNAME: 'node1'
  t2v-transformers:
      image:
semitechnologies/transformers-inference:sentence-transformers-paraphrase-mu
ltilingual-MiniLM-L12-v2
      deploy:
      resources:
      reservations:
```

```
            devices:
            - capabilities:
                  - 'gpu'
  qna-transformers:
      image: qna-module:latest
      deploy:
      resources:
      reservations:
            devices:
            - capabilities:
                  - 'gpu'
...
```

4.3 Download del dataset

Prima di poter utilizzare il sistema è necessario effettuare il download dei dataset:

1. Scaricare il file `ted_talks_it.csv` (41MiB) contenente le trascrizioni in italiano dei Ted Talk e posizionarlo all'interno della cartella `dataset/`
   https://www.kaggle.com/datasets/miguelcorraljr/ted-ultimate-dataset
2. Scaricare le tracce audio dei talk (21GiB) ed estrarre i file nella cartella `dataset/` in modo che sia presente il path `dataset/AUDIO/`
   https://www.kaggle.com/datasets/thegupta/ted-talks-audio

4.4 Caricamento dei dati

Il primo step per poter utilizzare il sistema consiste nel lanciare l'istanza di weaviate attraverso lo script `weaviate_start.sh`

Al primo avvio sarà necessario inizializzare il sistema attraverso il file python `system_init.py` . Eseguire dunque:

```
$ python system_init.py
```

Questo script si occupa di:

1. Creare lo schema
2. Creare gli oggetti
3. Creare le relazioni tra gli oggetti
4. Estrarre e caricare gli embedding audio

Innanzitutto viene estratto l'archivio *ted_talks_it.zip* ottenendo il file *ted_talks_it.csv.* L'archivio è quello già caricato sulla repository di GitHub*:*

```
if not exists("dataset/ted_talks_it.csv"):
     extract("dataset/ted_talks_it.zip")
```

Dopodiché si connette al server Weaviate (in locale), configurando anche le opzioni di batch: il `batch_size` andrebbe scelto in base alla configurazione hardware della macchina su cui si esegue il sistema. Per massimizzare la compatibilità è stato lasciato il valore 1 come default.

```
client = weaviate.Client("http://localhost:8080")
client.batch.configure(
   batch_size=1,
   dynamic=True,
   num_workers=16
)
```

Poi, se il database non risulta già configurato, procede a creare lo schema sulla base di un documento JSON simile a quello del Paragrafo 2.1:

```
def create_schema(ted_talk_object_schema):
   client.schema.create(ted_talk_object_schema)
```

Infine vengono creati gli oggetti:

```
batch.add_data_object(
   data_object=talk,
   class_name="TedTalk",
   uuid=id_to_uuid[talk["talk_id"]]
)
```

e le relazioni tra di essi:

```
batch.add_reference(
   from_object_uuid=this_talk_id,
   from_object_class_name="TedTalk",
   from_property_name="related_talks",
   to_object_uuid=related_talk_uuid,
   to_object_class_name="TedTalk",
)
```

Gli embedding degli oggetti vengono creati direttamente dal dbms in fase di caricamento degli oggetti stessi, secondo quanto specificato nello schema.

La variabile `id_to_uuid` è un vocabolario che serve per mappare ciascun id (di un ted talk nel CSV) nell'UUID di Weaviate. Questo è necessario per garantire la consistenza delle cross-reference. Si noti che, in generale, le cross-reference non sono bidirezionali: viene specificato il nodo di partenza con la keyword `from` e quello di destinazione con la keyword `to`. La relazione quindi, in generale, non è simmetrica.

Nota: il dataset da noi utilizzato non è completo, ci sono cioè dei riferimenti ad entry non presenti nel dataset stesso. Questi riferimenti sono stati ignorati in fase di caricamento nella base dati.

Infine vengono estratti gli embedding audio e caricati fornendo direttamente il vettore al dbms:

```
file_features =
audio_feature_extractor.extract_long_audio_embedding(audio_fi
le_path)
# Salva un data object
batch.add_data_object(
   data_object=talk_audio_object,
   class_name=TedTalkAudioClassName,
   uuid=talk_audio_uuid,
   vector=file_features
)

# Inserisce un riferimento da questo audio a quello correlato
batch.add_reference(
   from_object_uuid=talk_audio_uuid,
   from_object_class_name=TedTalkAudioClassName,
   from_property_name="talk_entry",
   to_object_uuid=talk_uuid,
   to_object_class_name=TedTalkClassName,
)
```

Il calcolo degli embedding audio è implementato nella classe `AudioFeatureExtractor` secondo le modalità descritte nel paragrafo 3.4

5. Modulo di query

Il modulo di query, implementato nel file *main.py*, consente ad un end-user di utilizzare l'applicativo per interrogare il database ed ottenere dei sample similari rispetto ad un prompt. Dei sample restituiti ne viene poi fatto un riassunto utilizzando il modello facebook/bart-large-cnn.

Innanzitutto viene istanziato `summarizer`, il modello per il riassunto di un testo, utilizzando la libreria Hugging Face Transformers, che fornisce implementazioni pre-addestrate di vari modelli di machine learning per il trattamento del linguaggio naturale.

```
summarizer = pipeline("summarization",
model=summarizer_model_name, device=device)
```

Poi occorre istanziare il client specificando l'URL dell'istanza di Weaviate.

```
client = weaviate.Client("http://localhost:8080")
```

Infine, attraverso un ciclo infinito, si può interagire con il sistema scegliendo tra le casistiche messe a disposizione.

```
while True:
   choices = ["Ricerca semantica", "Ricerca ibrida
testuale/semantica", "Question & Answer", "Ricerca audio",
"Quit"]
   index, _ = ask_user_choice("Cosa vuoi fare?", choices)

   if index == 0:
       semantic_search(client)
   elif index == 1:
       hybrid_search(client)
   elif index == 2:
       question_and_answer(client)
```

```
    elif index == 3:
        audio_search(client, device)
    elif index == 4:
        exit()
```

La funzione `semantic_search` prende in input il `client` per la connessione a Weaviate. La funzione procede chiedendo all'utente un testo da cercare e costruisce di conseguenza la query da sottoporre dal database, specificando nel campo `limit` il numero massimo di oggetti da recuperare. Il dbms restituirà ordinerà automaticamente i risultati restituendo solo quelli più pertinenti. Infine i risultati vengono presentati all'utente mediante la funzione `print_result`.

```
def semantic_search(client):
    text = input("Cosa cerchi?")
    query = build_query(client, limit=3)
    query = query.with_near_text({
        "concepts": [text]
    })
    results = execute_query(query)
    print("Ecco cosa ho trovato:")
    for talk in results:
        print_result(talk)
```

La funzione `build_query` definisce la `class_name` da cercare nel database e i `parameters` da restituire in output. Si definiscono poi degli `additional_parameters` che vogliamo in output, come l'uuid della entry e la misura di distanza dal prompt fornito.

La keyword `with_near_text` serve per implementare la ricerca semantica rispetto al prompt dell'utente. È possibile, alternativamente, cercare entry vicine alle altre oppure vicine ad un embedding fornito dall'utente.

La ricerca ibrida è implementata in modo del tutto analogo, ma al posto della `with_near_text` si usa una ricerca ibrida sui soli campi specificati (in questo caso la trascrizione della conferenza).

```
query = query.with_hybrid(query=text,
properties=["transcript"]) # Ricerca ibrida sulla
trascrizione
```

Il modulo di question & answer è implementato in modo simile:

```
user_provided_question = input("Fai una domanda:")

additional_parameters = "answer {hasAnswer certainty
property result startPosition endPosition}"
ask_details = {
   "question": user_provided_question,
   "properties": ["transcript"]
}

query = build_query(client, limit=3,
additional_parameters=additional_parameters)
query = query.with_ask(ask_details)  # ricerca ibrida sulla
trascrizione
results = execute_query(query)
```

Mentre la ricerca tramite audio procede prima con l'estrazione degli embedding, facendo poi una ricerca di tipo *Near Vector* navigando la proprietà `talk_entry` in quella che è in tutto e per tutto una giunzione:

```
print("Sto estraendo le feature audio...")
audio_features =
audio_feature_extractor.extract_long_audio_embedding(audio_
file_path)

print("Sto interrogando il database...")
parameters_string = ' '.join(parameters)
response = (
    client.query
    .get("TedTalkAudio", [
        "talk_entry { ... on TedTalk { " + parameters_string
+ " } }"
    ])
    .with_near_vector({
        "vector": audio_features
    })
    .with_limit(3)  # Limite la ricerca ai primi 3 oggetti
    .do()
)
```

5.1 Summarizer

Come descritto nel paragrafo 3.2.3, il modello BART utilizzato come summarizer ha un limite sulla lunghezza massima del testo da riassumere pari a 1024 token. Le trascrizioni dei talk possono però facilmente superare questa lunghezza, rendendo così necessario spezzare le trascrizioni in più parti e sintetizzare ognuna di queste parti, per poi concatenare i riassunti. La divisione del testo in segmenti deve essere eseguita facendo attenzione a non troncare le frasi.

La divisione di un testo in segmenti si basa sulla libreria nltk, usando un sentence tokenizer per separare le frasi.

```
sentences = nltk.tokenize.sent_tokenize(long_text,
language="italian")
```

Le frasi vengono poi nuovamente unite in segmenti lunghi non più di 1024 token.

I segmenti (o chunk) così generati vengono poi riassunti, utilizzando una beam search con 4 beam.

```
summaries = summarizer(text_chunks, length_penalty=5.0,
num_beams=4, max_length=256, early_stopping=True,
do_sample=False)
```

Il testo generato viene concatenato usando un singolo spazio come separatore.

```
summary = ""
for r in summaries:
   summary += r["summary_text"] + " "
```

6. Demo

In questo capitolo viene mostrata una demo dell'app.

Innanzitutto occorre inizializzare il sistema:
`$ python system_init.py`

L'output sarà:

```
Loading model facebook/wav2vec2-base-100k-voxpopuli...
Connecting to weaviate...
Getting the schema...

Meriche disponibili:
 1) cosine
 2) dot
 3) l2
 4) hamming
 5) manhattan
> Quale metrica di similarità usare? 1

Reading CSV...
Creating database schema...
Preparing data objects...
Storing objects...
Creating object references...
Talk id 31459 not found in this dataset. Reference dropped
...
Audio file dataset/AUDIO/61579.mp3 not found: ignoring this entry

Task completed.

Process finished with exit code 0
```

Dopodiché è possibile utilizzare il sistema per effettuare delle ricerche lanciando il file main.py

```
$ python main.py
```

L'output sarà:

```
Initializing summarizer model..

[nltk_data] Downloading package punkt to /home/ivano/nltk_data...
[nltk_data] Package punkt is already up-to-date!

Connecting to weaviate...

Cosa vuoi fare?
1) Ricerca semantica
2) Ricerca ibrida testuale/semantica
3) Question & Answer
4) Ricerca audio
5) Quit
> 1
> Cosa cerchi? Conferenze che parlando di sostenibilità ambientale

Ecco cosa ho trovato:
==========================================
# Talk id: 27793
# Title: 100 soluzioni per il cambiamento climatico
# Speaker 1: Chad Frischmann @ We the Future
# Native language: en
# Duration: 0:17:01
# Description: Cosa succederebbe se eliminassimo più gas serra di quanto ne
emettiamo nell'atmosfera? Questo scenario ipotetico, chiamato "riduzione", è
la nostra unica speranza per prevenire il disastro climatico, così dice lo
strategista Chad Frischmann. In questa conferenza progressista, condivide
soluzioni che esistono oggi per il cambiamento climaticoi; tattiche
convenzionali come l'uso dell'energia rinnovabile e migliore gestione del
suolo, così come alcuni approcci meno conosciuti, come cambiamenti nella
produzione alimentare, migliore pianificazione della famiglia e l'educazione
delle ragazze. Impara di più su come possiamo invertire il riscaldamento
globale e creare un mondo dove la rigenerazione, e non la distruzione, è la
regola.
```

URL:
https://www.ted.com/talks/chad_frischmann_100_solutions_to_reverse_global_warming/
TLDR: Il drawdown è un nuovo modo di pensare e agire sul riscaldamento globale. È quel punto quando eliminiamo più gas serra di quanto ne emettiamo nell'atmosfera. Queste sono soluzioni attuabili, scalabili e finanziariamente fattibili. Venti sono novità in arrivo. Otto delle prime 20 soluzioni riguardano il sistema alimentare. L'impatto del clima sugli alimenti potrebbe sorprendere molti. Proteggere foreste e zone umide protegge, amplia e crea nuovi serbatoi per il carbonio. L'analisi statistica permettono di adottare scelte ragionevoli per tutti gli imput usati in tutti i modelli. Abbiamo scelto un approccio conservativo, che è alla base di tutto il progetto. Tutti quei dati vengono inseriti nel modello, ambiziosamente ma plausibilmente proiettati nel futuro. Il cibo non viene sprecato nei paesi a basso reddito che lottano per sfamare la popolazione. Una dieta ricca di vegetali diventa la quarta soluzione per invertire il riscaldamento globale. Dobbiamo guardare tutta la catena produttiva e trovare perdite e sprechi. L'elettricità rinnovabile offre a tutti un accesso pulito e ampio all'energia. Una dieta ricca di vegetali, una riduzione degli sprechi garantiscono salute e cibo per tutti.

==========
Talk id: 30297
Title: È possibile combattere il riscaldamento globale? Partiamo dall'esperienza di come abbiamo protetto lo strato d'ozono.
Speaker 1: Sean Davis @ TEDxBoulder
Native language: en
Duration: 0:09:50
Description: Il protocollo di Montreal ha dimostrato che, insieme, possiamo agire per combattere i cambiamenti climatici. A trent'anni dalla firma del trattato ambientale di maggior successo al mondo, lo scienziato Sean Davis si interroga su ciò che è stato evitato con il divieto dell'utilizzo dei clorofluorocarburi e condivide utili insegnamenti per affrontare la crisi climatica che stiamo vivendo.
URL:
https://www.ted.com/talks/sean_davis_can_we_solve_global_warming_lessons_from_how_we_protected_the_ozone_layer/
TLDR: I CFC, i clorofluorocarburi, sostanze chimiche artificiali utilizzate come propellente per le bombolette spray. Lo strato di ozono costituisce la protezione solare della superficie terrestre, ed è fragile. Filtra oltre il 90% delle radiazioni nocive UV provenienti dal sole. In realtà,

paradossalmente, è stata la sicurezza umana che, al principio, ha determinato l'invenzione dei CFC. Nel 1970, gli scienziati capirono che i CFC avrebbero distrutto l'atmosfera e danneggiato lo strato di ozono. Questa scoperta ha scosso l'interesse pubblico. ha portato al divieto dell'uso di CFC per le bombolette spray negli Stati Uniti. Il Protocollo di Montreal ha evitato un cambiamento catastrofico ai danni dell'ambiente. Intorno al 2030 eviteremo milioni di casi di cancro della pelle all'anno bloccando così una crescita esponenziale. Facciamo sì that il meglio non sia nemico del bene, riflettiamo sul mondo che abbiamo evitated.

==========

Talk id: 1850
Title: Vendiamo una sostenibilità totale
Speaker 1: Steve Howard @ TEDGlobal 2013
Native language: en
Duration: 0:13:18
Description: I grandi edifici blu di Ikea hanno fatto spuntare pannelli solari e turbine eoliche; all'interno, gli scaffali sono pieni di lampadine al LED e cotone riciclato. Perché? Perché come dice Steve Howard: "La sostenibilità è passata dall'essere una cosa bella da fare a un dovere." Howard, responsabile della sostenibilità nel megastore di arredamento, parla del suo tentativo di vendere materiali e pratiche ecologiche, sia internamente che a clienti di tutto il mondo e lancia la sfida a altri giganti globali.
URL:
https://www.ted.com/talks/steve_howard_let_s_go_all_in_on_selling_sustainability/
TLDR: The Climate Group è un ONG sul cambiamento climatico. Ho lavorato sullo sviluppo e su questioni agricole nel sistema delle Nazioni Unite. Voglio massimizzare il mio impatto personale nel mondo. Abbiamo strategia di sostenibilità. Oggi abbiamo luci che producono luce e un po' di calore di contorno. Con un LED si risparmia l'85 per cento dell'elettricità consumata. Le lampadine classiche, quelle a incandescenza, hanno ancora a casa, a sprecare energia ogni volta. "Prendete l'obiettivo del 100 per cento" in silvicoltura. "Aumentano i raccolti, e si dimezzano i costi. Gli agricoltori passano oltre la soglia di povertà" IKEA ha sviluppato 300,000 pannelli solari, 14 parchi eolici di proprietà in sei paesi. Abbiamo installato 80 revisori nel mondo che assicurano buone condizioni di lavoro, proteggano i diritti umani. Entro il 2020, produrremo più energia rinnovabile di quanta. IKEA ha una fondazione impegnata a migliorare le vite e proteggere i diritti di 100 milioni di bambini entro il 2015. Abbiamo lanciato una rete aperta per

```
le donne questa settimana in IKEA, e faremo tutto il necessario per guidare il
cambiamento.
```

Facciamo un esempio con il question & answer:

```
Cosa vuoi fare?
1) Ricerca semantica
2) Ricerca ibrida testuale/semantica
3) Question & Answer
4) Ricerca audio
5) Quit
> 3
> Fai una domanda: Modi per ridurre la CO2

Ecco cosa ho trovato:
# Risposta: un aumento graduale della carbon tax
# Certezza: 0.3749667167663574
# Estratto dal talk 2784: 'Una soluzione climatica dove tutti possono
vincere'
```

7. Sviluppi futuri

In futuro si potrebbe pensare di cambiare modello di linguaggio, aggiungere un modulo di Automatic Speech Recognition per consentire all'utente di interagire col sistema non solo per via testuale ma anche per via vocale oppure dare maggiore flessibilità all'utente rispetto alla metrica di similarità da utilizzare. Si potrebbe pensare di far migrare il servizio da localhost al cloud (AWS oppure Azure), oppure cambiare tecnologia (da Weaviate a Milvus oppure QDrant). Si può pensare di sviluppare un'interfaccia grafica per rendere tutto più piacevole da usare. Inoltre sarebbe interessante analizzare il comportamento di ciascuna metrica di similarità rispetto alla ricerca, quindi valutare le prestazioni sui related talk del dataset. Infine un fine-tuning dei modelli potrebbe migliorare notevolmente le performance del sistema.

8. Conclusioni

I moderni DBMS vettoriali, come Weaviate, permettono interrogazioni potentissime e in tempi anche molto contenuti. La ricerca semantica basata su embedding permette di indicizzare miliardi (*sic!*) di data object in maniera efficiente. Quella generativa, d'altro canto, migliora i risultati di ricerca sfruttando i Large Language Models (LLM) come GPT-3 per creare esperienze di ricerca di nuova generazione, interrogando in maniera davvero libera un'intera base di conoscenza. Quella ibrida combina tecniche tra loro per un'esperienza allo stato dell'arte. Weaviate ci ha particolarmente colpito per le sue funzionalità, per le prestazioni e per la chiarezza della documentazione. La semplicità con cui permette di integrare varie tecnologie tra loro, a partire dai LLM fino agli embedding passando per l'hosting su cloud, lo rendono a nostro avviso un'ottima scelta per questo caso d'uso. D'altro canto Python, tra i linguaggi di scripting più usati nel 2023, ha permesso tutto questo in poche righe di codice aumentando, conseguentemente, la leggibilità e la manutenibilità degli artefatti.

Bibliografia

[WEA]: Weaviate. https://weaviate.io/

[HUG]: Hugging face. https://huggingface.co/

[SLP]: Dan Jurafsky and James H. Martin, Speech and Language Processing (3rd edition draft). https://web.stanford.edu/~jurafsky/slp3/

[M1]: Reimers, Nils and Gurevych, & Iryna. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* (Vol. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing). Association for Computational Linguistics. arXiv:1908.10084

[M2]: M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. arXiv:1910.13461*

[M3]: Pengcheng He, Jianfeng Gao, Weizhu Chen (2023). *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*. arXiv:2111.09543

[M4]: Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. arXiv:2003.10555

[M5] Yu-An Chung, J. Glass (2018) Speech2Vec: *A Sequence-to-Sequence Framework for Learning Word Embeddings from Speech* arXiv:1803.08976

[CON] Chen, G. (2022). *Homophone Reveals the Truth: A Reality Check for Speech2Vec*. arXiv:2209.10791

[VOX] Wang, C., Rivière, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., Williamson, M., Pino, J., & Dupoux, E. (2021). *VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation*. arXiv:2101.00390