

# Relazione tirocinio presso “Nexa Center for Internet & Society”

Tema:

Applicazione modelli di machine learning su Knowledge Graph  
per sviluppo sistema di Named-Entity Disambiguation



**Politecnico  
di Torino**

Vito Perrucci

Matricola: 283068

Anno accademico 2022/2023

# Introduzione

Il Politecnico di Torino offre la possibilità di effettuare un tirocinio curriculare al terzo anno del percorso di laurea triennale. Se lo studente sceglie di intraprendere questa attività, è messo davanti a due ulteriori scelte: il quantitativo di ore da dedicare al progetto (nel mio percorso di laurea vi erano due possibili opzioni: 250 o 300 ore) e l'azienda all'interno della quale spendere le proprie ore per l'ottenimento dei crediti curriculari.

Sulla base alle mie prospettive future, ho scelto di svolgere il tirocinio presso un centro di ricerca, dove poter sperimentare cosa succede all'interno di un gruppo dedito alla scoperta di nuove tecniche di programmazione che possano collaudare idee di sviluppo tramite lo studio approfondito dell'immenso mondo dell'Internet.

Questo è proprio quello di cui il Centro Nexa si occupa e affrontare questo tirocinio mi ha fatto appassionare ancora di più al mondo della programmazione; relazionandomi con il team, ho imparato che è importante cimentarsi con argomenti al di sopra della propria preparazione, che non bisogna avere timore della loro complessità ma anzi è fondamentale aver voglia di conoscere e imparare, così da realizzare che abbiamo diverse possibilità per affrontare un nuovo concetto, una nuova realtà, una nuova metodologia di studio che, per forza di cose, bisogna intraprendere per portarsi avanti con l'apprendimento.

Mi presento,

mi chiamo Vito Perrucci e sono uno studente di ingegneria informatica presso il politecnico di Torino. Ho svolto il tirocinio in questione durante il secondo semestre del terzo anno; per la precisione ho iniziato il 23 marzo 2023, e concluso le mie 250 ore il 12 giugno 2023. Il percorso è stato svolto durante altri tre corsi e lo stesso progetto di tirocinio è stato utile per far sì che riuscissi a mettermi alla prova con una ulteriore esperienza, diversa dal "classico" svolgimento delle ore di studio.

Insomma, un'esperienza fantastica che ho consigliato a tanti amici, e che sicuramente continuerò a consigliare se ne avrò l'occasione.

# Tirocinio

Le prime cento ore circa del percorso sono state dedicate allo studio autonomo di concetti fondamentali, utili per la corretta comprensione delle successive attività.

Lo studio ha girato intorno al concetto di NLP (Natural Language Processing), un campo dell'intelligenza artificiale (IA) che si occupa di consentire alle macchine di comprendere, interpretare e generare il linguaggio umano in modo naturale. In altre parole, l'NLP mira a far sì che le macchine possano "capire" il nostro linguaggio parlato o scritto e rispondere ad esso in modo significativo.

Immaginiamo ad esempio di parlare con un assistente virtuale, come Siri o Alexa. L'NLP consente a questi assistenti di comprendere ciò che stiamo chiedendo e fornire una risposta coerente. Questo è possibile grazie a complessi algoritmi e modelli di linguaggio che analizzano le parole, la struttura delle frasi e il contesto per arrivare a una comprensione accurata.

Ci sono diverse sfide nell'NLP, poiché il linguaggio umano è molto complesso e spesso ambiguo. Le parole possono avere significati multipli a seconda del contesto, e la sintassi può variare notevolmente da una lingua all'altra oltre che all'interno della stessa lingua.

Le applicazioni dell'NLP sono molteplici e stanno diventando sempre più presenti nella vita quotidiana. Oltre agli assistenti vocali, l'NLP è utilizzato nei motori di ricerca per fornire risultati di ricerca pertinenti, nei sistemi di traduzione automatica, nell'analisi dei sentimenti per comprendere le scelte e gli orientamenti dell'opinione pubblica, nella classificazione automatica di testi, nella creazione di chatbot, e molto altro.

In sintesi, l'NLP è una tecnologia straordinaria che rende possibile la comunicazione tra esseri umani e macchine in modo più naturale ed efficace, contribuendo a rendere il mondo digitale sempre più interattivo e accessibile.

Nelle mie ore di studio mi è sembrato opportuno, prima di poter entrare nel dettaglio con l'argomento, approfondire alcuni concetti generali.

## Intelligenza Artificiale

L'Intelligenza Artificiale è un campo dell'informatica che mira a creare sistemi o macchine in grado di eseguire attività che richiedono intelligenza umana. L'obiettivo principale dell'IA è replicare alcune delle capacità cognitive umane, come il ragionamento, la comprensione del linguaggio, la percezione visiva, l'apprendimento, la risoluzione di problemi e la presa di decisioni.

L'IA può essere divisa in due categorie: IA debole (o ristretta) e IA forte (o generale). L'IA debole si concentra su compiti specifici e limitati, come il riconoscimento vocale o il gioco degli scacchi. L'IA forte, invece, mira a emulare l'intelligenza generale e l'autonomia paragonabile a quella umana.

È importantissimo specificare che a volte il termine 'intelligenza' può far pensare che un sistema informatico possa essere davvero scaltro e quindi essere in grado di ragionare, avere intuizioni o, in parole povere, collegare i puntini per trovare la soluzione a un problema proposto.

In alcuni ambiti questo potrebbe quasi essere raggiunto, ma non perché riusciamo a far ragionare le macchine. Non siamo in grado di creare dei cervelli, ma siamo in grado di allenare le macchine ad utilizzare enormi database di informazioni ai fini di poter collegare i vari dati e produrre una risposta (la risposta non è per forza letterale) che riesce ad agevolare di gran lunga dei compiti umani.

Ho letto molto, mosso dalle necessità didattiche ma anche con una certa e forte curiosità, articoli riguardanti il timore che l'IA possa rappresentare un problema per la società odierna.

L'idea che le macchine possano soppiantare gli esseri umani nei posti di lavoro, già fortemente scarsi, è fonte di preoccupazione per molti. Tuttavia, è importante sottolineare che l'intelligenza artificiale deve essere utilizzata in modo oculato e responsabile. La sostituzione completa dell'uomo con le macchine non è una prospettiva realistica.

Al contrario, l'intelligenza artificiale ha il potenziale per agire come un valido alleato dell'umanità. Le macchine possono svolgere compiti ripetitivi e meccanici con una precisione senza precedenti e in tempi più brevi, evitando che siano completamente a carico degli esseri umani. Ciò permette alle persone di concentrarsi su attività più creative, complesse e significative.

Prima ho parlato di un vero e proprio allenamento (più avanti ne parlerò con il nome di apprendimento) sottoposto alle macchine per poter essere pronte ad affrontare particolari compiti. Mi riferisco al processo di Machine Learning, del quale ho studiato meccanismi e tecniche di approccio.

Durante il tirocinio ho seguito un articolo pubblicato da un ragazzo di nome Fabio Chiusano che spiega alcuni fondamenti di programmazione per la realizzazione di una 'knowledge base'. In sostanza si tratta di una particolare metodologia di conservazione delle informazioni attraverso la quale è possibile accedervi creando collegamenti tra le 'parole' e comprendere determinati concetti. Di questo, però, aggiungo solo un approfondimento a fine lavoro; prima ci tengo a specificare qualcosa su altri concetti importanti ai fini della comprensione.

## **Machine Learning**

Il Machine Learning è una sotto disciplina dell'IA che si basa sulla costruzione di algoritmi e modelli statistici per permettere alle macchine di "imparare" dai dati e migliorare le prestazioni dei compiti senza essere esplicitamente programmate.

A differenza della programmazione tradizionale, in cui le istruzioni sono codificate manualmente, il Machine Learning utilizza dati di addestramento per generare modelli che generalizzano dai dati stessi, consentendo alle macchine di prendere decisioni o effettuare previsioni basate su nuove informazioni.

Ci sono diversi tipi di apprendimento automatico, tra cui l'apprendimento supervisionato (dove il modello viene addestrato utilizzando esempi di input e output corrispondenti), l'apprendimento non supervisionato (dove il modello cerca di scoprire pattern nei dati senza etichette) e l'apprendimento per rinforzo (dove il modello impara tramite riscontri positivi o negativi in base alle sue azioni).

In breve, l'IA si concentra sull'obiettivo di replicare l'intelligenza umana in varie forme, mentre il Machine Learning è una tecnica specifica nell'ambito dell'IA che consente alle macchine di imparare dai dati e migliorare le prestazioni nei compiti assegnati.

Spesso il Machine Learning viene utilizzato come strumento fondamentale per realizzare applicazioni e sistemi di Intelligenza Artificiale più ampi.

## Named Entity Recognition (NER)

Il Named Entity Recognition è una tecnica che permette di individuare e classificare le "entità nominate" all'interno di un testo. Le entità nominate sono sostanzialmente nomi di persone, luoghi, organizzazioni, date e altre informazioni specifiche e identificabili all'interno di una frase o di un documento.

Per esempio, consideriamo la seguente frase: "Vito studia presso il Centro Nexa a Torino dal 2023". L'obiettivo del NER è estrarre le seguenti entità nominate:

- Persona: "Vito"
- Organizzazione: "Centro Nexa"
- Luogo: "Torino"
- Data: "2023"

L'implementazione di codice Python per il NER consiste nell'utilizzo di modelli di apprendimento automatico (machine learning) che hanno appreso da esempi come identificare queste diverse tipologie di entità. Una volta addestrato, il modello può essere utilizzato per analizzare testi nuovi e rilevare le entità nominate in essi.

## Named Entity Disambiguation (NED)

Il Named Entity Disambiguation è una tecnica più avanzata rispetto al NER. Quando il NER identifica le entità nominate, potrebbe capitare che alcune di esse abbiano più di una possibile interpretazione. Ad esempio, il nome "Apple" potrebbe riferirsi sia alla società di tecnologia che produce iPhone e Mac, sia al frutto.

La NED si occupa di risolvere queste ambiguità e associare un'unica interpretazione corretta a ciascuna entità. In altre parole, la NED "disambigua" l'entità, scegliendo l'interpretazione più appropriata in base al contesto.

Per esempio, consideriamo la seguente frase: “Vito canta ma non ha acceso il gelato”. La NED dovrebbe capire dal contesto che il termine “gelato” si riferisce al microfono (non al dolce) e riesce a farlo analizzando il contesto suggerito dalle parole di contorno. È chiaro che i termini “**canta**” e “**acceso**” hanno molte combinazioni con gelato solo quando esso ci si riferisce al microfono.

L'implementazione di codice Python per la NED può coinvolgere approcci più sofisticati, come l'utilizzo di conoscenza esterna, disambiguazione basata su contesto e/o la combinazione con altri metodi di analisi del linguaggio naturale.

Una volta chiariti questi concetti, ho cominciato a studiare il funzionamento di librerie, frameworks o singole funzioni di python per poter lavorare autonomamente con il linguaggio e implementare algoritmi che potessero testare la corretta comprensione degli argomenti studiati.

Ci sono state, quindi, ulteriori ore di studio; posso dire con certezza che non ho mai smesso di studiare durante l'intero percorso.

Nella fase finale mi sono dedicato alla stesura di un codice che mi ha messo molto alla prova e ha coperto buona parte delle ore del progetto; vediamo di che si tratta.

# Progetto

L'obiettivo è quello di estrapolare, da un testo, una lista di entità tramite modelli pre-addestrati di NER e successivamente servirsi della libreria python 'wikipedia' per poter disambiguare le varie entità e produrre l'URI corrispondente, ovvero il link della pagina wikipedia relativa all'entità disambiguata.

Le prime istruzioni di codice sono riservate all'inserimento delle dichiarazioni degli import e alla configurazione della libreria di Wikipedia in lingua italiana.

```
#per NER
from transformers import AutoTokenizer, AutoModel
import spacy

#per embedding e similarità
import torch
import numpy
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import util

#per ricerca su wikipedia
import wikipedia
wikipedia.set_lang('it')
from wikipedia.exceptions import DisambiguationError
```



Si può notare che il codice è strutturato in diverse sezioni. La prima sezione riguarda l'Identificazione delle Entità Nominative (Named-Entity Recognition), che si basa su rilevatori di Token e su modelli come quello di Spacy.

Successivamente, vi sono le dichiarazioni delle librerie che consentono l'embedding del testo, una tecnica innovativa utilizzata per la Disambiguazione delle Entità Nominative (Named-Entity Disambiguation).

Infine, è necessario l'utilizzo della libreria 'wikipedia', che permette di cercare determinati URI nel dominio o di generarli per essere salvati.

È importante sottolineare che la libreria viene configurata per la lingua italiana ('it'), poiché l'intero codice si basa su modelli addestrati con testi italiani. Anche se potrebbero emergere delle problematiche nel tentativo di eseguire il programma con testi in inglese, queste potrebbero essere affrontate cambiando i modelli pre-addestrati (ce ne sono diversi altamente funzionali) e impostando la lingua di Wikipedia su 'en'.

```
entities = get_entities(text)
print("Dato il seguente testo:\n", text, "\n\nLe entità trovate sono:\n")
for entity in entities:
    print(entity)
```

Il programma inizialmente estrae le entità dal testo e per farlo si serve della funzione **get\_entities()**

```
def get_entities(text):
    nlp = spacy.load("it_core_news_sm")
    doc = nlp(text)
    entities = []
    for entity in doc.ents:
        if entity.text not in entities:
            entities.append(entity.text)
    return entities
```

La funzione accetta come parametro un testo, e tramite la libreria Spacy carica un modello all'interno della variabile 'nlp'.

Questa variabile si occupa di fare NER sul testo e, tramite **.ents** , è possibile ottenere tutte le entità estratte, che vengono inserite in un vettore 'entities'

Una volta ottenute le entità possiamo quindi passare alla seconda fase, e cioè comprendere di quali entità stiamo parlando. Questo processo è gestito dalla funzione **get\_wikipedia\_pages()** che vuole come parametri la lista di entità appena trovate e il testo di partenza

`get_wikipedia_pages(entities, text)`

```
def get_wikipedia_pages(entities, text):
    summaries = {}
    for entity in entities:
        search_results = wikipedia.search(entity, results = 5)

        if len(search_results) > 0:
            bestDis = calcola_migliore_similarita(entity, search_results, text)
            result = search_results[bestDis]

            try:
                page = wikipedia.page(result)
                summaries[entity] = wikipedia.summary(result)
            #questo except serve perchè il risultato migliore può comunque generare disambiguation
            #nel caso dovesse succedere .options contiene le varie possibilità
            except wikipedia.exceptions.DisambiguationError as r:
                bestDis = calcola_migliore_similarita(entity, r.options, text)
                page = wikipedia.page(r.options[bestDis])
                summaries[entity] = wikipedia.summary(r.options[bestDis])

            print("\nEntity : " + entity + "\n Url:      " + page.url + "\nSummary page: " + summaries[entity] + "\n")

        else:
            print("Nessuna pagina wikipedia per l'entità  '" + entity + "'")

    return
```

Come funziona nel dettaglio?

Per ogni entità trovata, effettuiamo una ricerca su wikipedia tramite **wikipedia.search()**, e inseriamo i risultati in un vettore di stringhe.

Ovviamente wikipedia può non trovare nulla e in questo caso viene prodotto un messaggio di errore che indica che non è stata trovata nessuna pagina per l'entità. Nel caso in cui, invece, alcuni risultati sono stati prodotti, allora bisogna passare ad una tecnica che permette di capire quale dei risultati è quello corretto. Per farlo ho creato una funzione dal nome **calcola\_migliore\_similarita()** che restituisce l'indice della posizione della corretta disambiguazione nel vettore di risultati.

Come avviene il processo? In sostanza ogni risultato ha una sua pagina URI facilmente trovabile tramite la libreria wikipedia che, tramite il sottocomando **.summary()**, permette di ottenere una sorta di riassunto/sommario della pagina trovata.

Questo sommario viene confrontato con il testo di partenza per poter attribuire un 'punteggio di somiglianza' al risultato, e prendere quindi il risultato con il punteggio più alto.

Punteggio e somiglianza sono parole che per ora possono confondere le idee ma a breve sarà tutto più chiaro. Bisogna tenere bene a mente che sono proprio i numeri ad aiutarci a trovare delle soluzioni ottimali.

In questo caso sfrutteremo le trasformazioni vettoriali di testi e cercheremo di trovare la correlazione tra i vettori usando la somiglianza coseno, una tecnica che sfrutta l'angolo tra due vettori per quantificare quanto due vettori sono 'simili'.

## funzione `calcola_migliore_similarita()`

```
def calcola_migliore_similarita(entity, search_results, text):
    max_sim = 0
    best_entity = 0

    embedding1 = calcola_embedding_testo_entity(text, entity)
    embedding1 = numpy.array(embedding1).reshape(1, -1)

    link = generate_wikipedia_link(entity)

    for i,result in enumerate(search_results):
        try:
            wikilink = wikipedia.page(result).url
            if link == wikilink:
                return i;
        except wikipedia.exceptions.DisambiguationError:
            print ("pagina trovata ma disambiguazione, adesso parte il check della similarità")

        try:
            text2 = wikipedia.summary(result)
            embedding2 = calcola_embedding_testo_entity(text2, result)
            #embedding2 = calcola_embedding_testo(text2)

        except wikipedia.exceptions.DisambiguationError as r:
            print("l'entità " + entity + " ha diverse ambiguità:")
            #multipla = True
            for e in r.options:
                text2 = wikipedia.summary(e)
                embedding2 = calcola_embedding_testo_entity(text2, result)
                embedding2 = numpy.array(embedding2).reshape(1, -1)

                cosine_scores = cosine_similarity(embedding1, embedding2)[0][0]
                print("\t risultato: " + entity + " ---->ambiguità : " + e + " ----->sim: ", cosine_scores)
                if cosine_scores > max_sim:
                    max_sim = cosine_scores
                    best_entity = i

    cosine_scores = util.cos_sim(embedding1,embedding2)
    #if multipla == False:
        #print("entità: " + entity + " ---->search_result: " + result + " ----->sim: ", cosine_scores)
    #multipla = False

    if cosine_scores > max_sim:
        max_sim = cosine_scores
        best_entity = i
    return best_entity
```

Per capire meglio proviamo a sfruttare l'esempio già fatto:

Di quale 'gelato' stiamo parlando nella frase "Vito canta ma non ha acceso il gelato"?

Il programma rileva l'entità e nella lista di risultati ne aggiunge due possibili:

1. Gelato (dolce)
2. Gelato (microfono)

Per capire quale delle due è quella più corretta, il codice confronta la pagina wikipedia dei risultati con il testo di partenza ("Vito canta ma non ha acceso il gelato") e attribuisce un punteggio più alto al secondo risultato perché la pagina wikipedia contiene le parole 'accendere' e 'cantare' che invece la pagina del primo (Gelato - dolce) non ha.

Ma come è possibile assegnare un punteggio ad un risultato?

Si tratta di quello di cui parlavo prima quando ho introdotto il termine di somiglianza; la funzione **calcola\_migliore\_similarita()** confronta l'embedding del testo di partenza con l'embedding del summary di ogni risultato.

L'embedding di un testo è la sua rappresentazione vettoriale e viene generato dalla funzione **calcola\_embedding\_testo()**.

Questa funzione usa pyTorch, un framework particolarmente flessibile che consente di lavorare con il concetto di tensori, che sono strutture dati simili a vettori multidimensionali; il tutto è combinato all'aiuto di AutoTokenizer utilizzato per suddividere il testo in unità più piccole, come parole o sottostringhe, chiamate "token". Questo è fondamentale per preparare il testo alla conversione in una forma strutturata e numerica che può essere maneggiata più facilmente da un punto di vista di confronto.

Di seguito l'implementazione della funzione **calcola\_embedding\_testo()**.

```
def calcola_embedding_testo_entity(text, entity):
    tokenizer = AutoTokenizer.from_pretrained("obrizum/all-MiniLM-L6-v2")
    model = AutoModel.from_pretrained("obrizum/all-MiniLM-L6-v2")

    text_split = text.split('\n')
    embeddings = []
    for frase in text_split:
        if entity in frase:
            encoded_input = tokenizer(frase, padding=True, truncation=True, max_length=512, return_tensors='pt')
            with torch.no_grad():
                model_output = model(**encoded_input)
            sentence_embedding = torch.mean(model_output.last_hidden_state, dim=1).squeeze()
            embeddings.append(sentence_embedding)

    if len(embeddings) == 0:
        for frase in text_split:
            encoded_input = tokenizer(frase, padding=True, truncation=True, max_length=512, return_tensors='pt')
            with torch.no_grad():
                model_output = model(**encoded_input)
            sentence_embedding = torch.mean(model_output.last_hidden_state, dim=1).squeeze()
            embeddings.append(sentence_embedding)

    embedding_testo = torch.mean(torch.stack(embeddings), dim=0).numpy()
    return embedding_testo
```

Inizialmente questa funzione era più semplice; poi ho deciso di modificarla per renderla più funzionale.

In sostanza il testo che deve essere trasformato in vettore, viene prima spezzettato in frasi tramite **text\_split()**, poi ogni frase assume una forma numerica vettoriale tramite modelli pre-addestrati, e infine la lista degli embeddings viene trasformata in un unico embedding tramite una media vettoriale dei singoli vettori frase.

La modifica che è stata fatta per rendere più funzionale il codice riguarda l'aggiunta di una scelta precisa delle frasi da considerare.

Per far sì che il tutto funzioni meglio è necessario che non tutte le frasi contribuiscano alla creazione del vettore del testo ma che contribuiscano solo ed esclusivamente le frasi che contengono al loro interno la parola dell'entità di partenza.

Una volta ottenuti gli embeddings dei due testi è possibile utilizzare **cosine\_similarity()**, che accetta come parametri i due vettori e restituisce un valore compreso tra -1 e 1 che indica la somiglianza dei due vettori.

Se i vettori sono uguali restituisce 1, se sono estremamente diversi allora restituisce un numero molto prossimo allo 0. Se invece sono inversamente uguali (concetto che non ci serve in questo caso) allora il numero sarà più prossimo a -1.

Inutile specificare che per avere il risultato più corretto non ho fatto altro che prendere quello con il numero di similarità più prossimo a 1 e quindi più alto possibile.

# Esempio

Il testo di partenza è:

Fatto sta che Lollobrigida è il marito di Arianna Meloni, sorella della premier e quindi anche suo cognato, oltre che fedelissimo alleato in Fratelli d'Italia.

È importante considerare che Lollobrigida genera sicuramente più risultati in quanto ci sono diverse persone con questo cognome.

Inoltre, Arianna Meloni non è sufficientemente famosa da avere una pagina wikipedia, quindi quale sarà il suo URI?

Il codice produce il seguente risultato:

Le entità trovate sono:

Lollobrigida  
Arianna Meloni  
Fratelli d'Italia

Per ogni entità il programma trova il link della pagina wikipedia corrispondente all'entità. Ecco i risultati prodotti:

Entity : Lollobrigida

Url: [https://it.wikipedia.org/wiki/Francesco\\_Lollobrigida](https://it.wikipedia.org/wiki/Francesco_Lollobrigida)

Summary page: Francesco Lollobrigida (Tivoli, 21 marzo 1972) è un politico italiano, deputato dal 2018 di Fratelli d'Italia, partito per il quale è stato capogruppo per tutta la XVIII legislatura. Dal 22 ottobre 2022 è ministro dell'agricoltura, della sovranità alimentare e delle foreste nel governo Meloni.

Entity : Arianna Meloni

Url: [https://it.wikipedia.org/wiki/Francesco\\_Lollobrigida](https://it.wikipedia.org/wiki/Francesco_Lollobrigida)

Summary page: Francesco Lollobrigida (Tivoli, 21 marzo 1972) è un politico italiano, deputato dal 2018 di Fratelli d'Italia, partito per il quale è stato capogruppo per tutta la XVIII legislatura. Dal 22 ottobre 2022 è ministro dell'agricoltura, della sovranità alimentare e delle foreste nel governo Meloni.

Entity : Fratelli d'Italia

Url: [https://it.wikipedia.org/wiki/Fratelli\\_d%27Italia\\_\(partito\\_politico\)](https://it.wikipedia.org/wiki/Fratelli_d%27Italia_(partito_politico))

Summary page: Fratelli d'Italia (FdI) è un partito politico italiano di destra ed estrema destra fondato nel 2012 da Ignazio La Russa, Guido Crosetto e Giorgia Meloni, la quale lo presiede dal 2014.

È descritto come un partito nazional-conservatore, nazionalista, populista di destra, reazionario, conservatore in campo sociale, e post-fascista. In campo internazionale è euroscettico e atlantista. Derivato da una scissione del Popolo della Libertà, Fratelli d'Italia si presenta come il prosecutore ideale della tradizione politica di Alleanza Nazionale, partito di destra post-fascista, evoluzione del Movimento Sociale Italiano, partito di ispirazione neofascista fondato da ex-membri del disciolto PNF (1921-1943) e del PFR (1943-1945). Fino al 2017 comparivano nel simbolo anche il nome di Alleanza Nazionale e la sigla "MSI". Tutt'oggi il simbolo mantiene anche la fiamma tricolore, storicamente utilizzata dal MSI

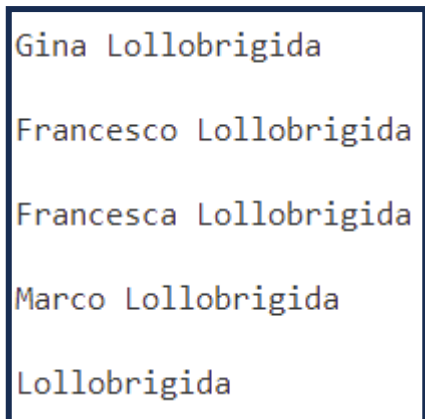


Le tre entità sono disambiguate correttamente.

Vediamole nel dettaglio:

- Lollobrigida:

i primi cinque possibili risultati sono i seguenti:

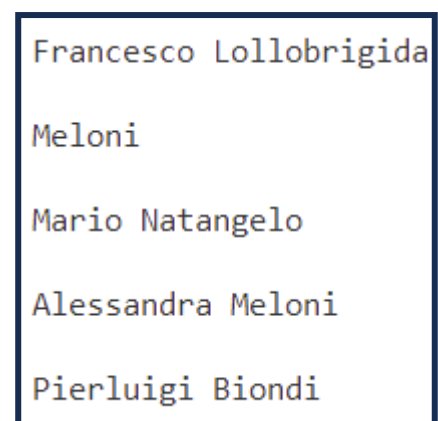


Gina Lollobrigida  
Francesco Lollobrigida  
Francesca Lollobrigida  
Marco Lollobrigida  
Lollobrigida

dall'entità viene prodotta la pagina di Francesco Lollobrigida, marito di Arianna Meloni, cognato della Premier e alleato di Fratelli d'Italia. Queste sono tutte informazioni reperibili dall'URI [https://it.wikipedia.org/Francesco Lollobrigida](https://it.wikipedia.org/Francesco_Lollobrigida).

- Arianna Meloni

I primi cinque possibili risultati sono i seguenti:



Francesco Lollobrigida  
Meloni  
Mario Natangelo  
Alessandra Meloni  
Pierluigi Biondi

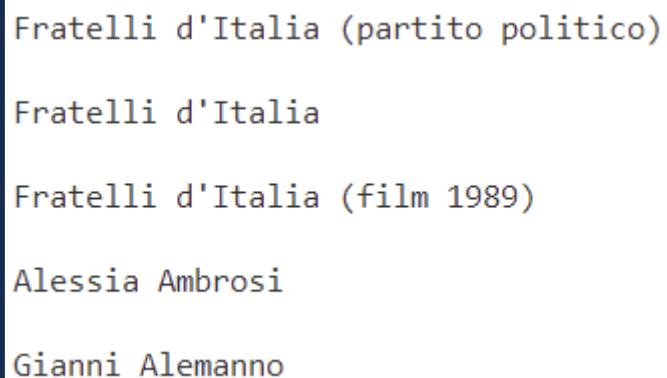
Come dicevo, non esiste una pagina wikipedia di Arianna Meloni e quindi le soluzioni più corrette sono quelle che in qualche modo possono condurre a lei.

Il Codice sceglie il marito e se questo accade è perché semplicemente ha più corrispondenze;

Si può immaginare, ad esempio, che nella pagina di Francesco viene citata la moglie, mentre quest'ultima non viene mai citata in quella della sorella (come si può notare, la pagina di Giorgia Meloni non è neanche nelle prime 5 ricerche).

- Fratelli d'Italia

I primi cinque possibili risultati sono i seguenti:



```
Fratelli d'Italia (partito politico)
Fratelli d'Italia
Fratelli d'Italia (film 1989)
Alessia Ambrosi
Gianni Alemanno
```

In questo caso ci sono pagine con lo stesso nome ma con etichette diverse. Ogni volta che questo è possibile esiste un risultato senza etichetta (il secondo) che comprende tutti i risultati etichettati e quindi comprenderà sia il partito politico che il film del 1989. Questo nel codice è contemplato e infatti ogni qual volta viene generata tra i risultati una pagina del genere (che prende il nome di pagina di disambiguazione), il codice analizza tutte le possibili accezioni dell'entità e conferma quella più corretta.

In questo caso quella più corretta è ovviamente il partito politico.

# Conclusione

La scrittura del codice si è rivelata una vera e propria sfida, nonostante avessi a disposizione strumenti potentissimi.

Le librerie, i framework e i modelli che ho utilizzato offrono possibilità sorprendenti, molte delle quali sono rimaste inesplorate, e talvolta mi sono trovato a dover studiare a fondo ciascuno di essi, anche solo per padroneggiarne uno.

Le difficoltà incontrate, le lunghe sessioni di lavoro e gli sforzi fatti hanno ricevuto una giusta ricompensa, poiché il bagaglio di conoscenze acquisite è diventato davvero inestimabile.

L'argomento che ho affrontato si è rivelato affascinante sin dalle prime fasi, e sono felice di aver preso l'iniziativa di tuffarmi in questa stimolante sfida.

# Approfondimento:

## KNOWLEDGE BASE FROM TEXT

Una Knowledge Base (KB) rappresenta un insieme organizzato di informazioni, pronte per essere utilizzate in analisi o inferenze. Ho acquisito una comprensione chiara di questo concetto grazie a un articolo scritto da Fabio Chiusano, il quale ha contribuito significativamente alla mia comprensione dell'applicazione dell'intelligenza artificiale.

Nel suo articolo, Fabio introduce il concetto di "triple", che rappresenta le relazioni tra diverse entità. Una tripla consiste di due entità e un terzo elemento che rappresenta la relazione tra le prime due. Un esempio chiarificatore è la tripla "Fabio vive in Italia", dove le entità sono Fabio e Italia, e la relazione che li lega è "vive in".

Immaginiamo di estrarre diverse triple da un testo. Queste triple possono essere organizzate in una struttura dati che conserva informazioni fondamentali, pronte per essere utilizzate e sviluppate ulteriormente. Nel suo articolo, Fabio fornisce una guida passo dopo passo per la creazione di una Knowledge Base, partendo da un breve testo e successivamente applicando lo stesso principio a un articolo più esteso.

Nel mio repository GitHub, ho condiviso un programma che implementa i principi illustrati da Fabio. Questo programma permette di generare una Knowledge Base da testi brevi, testi più lunghi o persino da pagine di notizie casualmente prese da Google tramite una funzione dedicata. Nel file README.md allegato, vi sono dettagliate spiegazioni sul funzionamento delle varie funzioni.

Per una visione completa e per esaminare il codice del progetto di tirocinio correlato, è possibile visitare il seguente link al mio repository GitHub:

<https://github.com/Vito-Perrucci>