

# Dataset Builder Ontology

Vincenzo Buttari, Francesco Ciavarella

8 febbraio 2025

# Indice

<b>1</b>	<b>Obiettivo del progetto</b>	<b>1</b>
<b>2</b>	<b>Ontologia</b>	<b>2</b>
2.1	Campi dell'ontologia . . . . .	3
<b>3</b>	<b>Funzionalità</b>	<b>4</b>
3.1	Modello predittivo . . . . .	4
3.2	Modello predittivo con Grid Search . . . . .	6
3.2.1	Cross-validation a 5 fold . . . . .	7
3.3	Confronto tra modello predittivo base e modello predittivo con Grid Search	8
3.4	Come funziona PyKEEN con TransE . . . . .	9
<b>4</b>	<b>Requisiti funzionali</b>	<b>12</b>
4.1	Prerequisiti . . . . .	12
4.1.1	Installazione di Java . . . . .	12
4.2	Struttura del Progetto . . . . .	14
<b>5</b>	<b>Interfacce Utente: CLI e Web App</b>	<b>15</b>
5.1	Interfaccia a Riga di Comando (CLI) . . . . .	15
5.1.1	Esempi di Esecuzione dal CLI . . . . .	15
5.2	Interfaccia Web . . . . .	21
<b>6</b>	<b>Conclusioni</b>	<b>27</b>

# **Dataset Builder Ontology**

**Nome:** [Vincenzo Buttari](#)

**GitHub:**

[https://github.com/zeltarave/dataset\\_builder\\_ontology](https://github.com/zeltarave/dataset_builder_ontology)

**Matricola:** [776274](#)

**Nome: Francesco Ciavarella**

**GitHub:**

[https://github.com/zeltarave/dataset\\_builder\\_ontology](https://github.com/zeltarave/dataset_builder_ontology)

**Matricola: 782015**

8 febbraio 2025

# Capitolo 1

## Obiettivo del progetto

Il progetto mira a esplorare la creazione automatica di un'ontologia partendo da un dataset di dati generati casualmente. L'idea è di sfruttare Faker, una libreria Python che permette di generare dati fintizi, e utilizzarla come base per costruire un'ontologia in grado di rappresentare e organizzare tali dati in modo semantico. Per generare l'ontologia usiamo la libreria Python OWLReady2. OWLReady2 è uno strumento potente e flessibile per chiunque desideri integrare ontologie e ragionamento logico grazie alla sua interfaccia orientata agli oggetti e alle capacità di inferenza. Questi dati possono poi essere utilizzati come base addestrare un'intelligenza artificiale. OWLReady2 è dotato di un potente reasoner chiamato HermiT attraverso la funzione `sync_reasoner()` che permette di eseguire un ragionamento completo sull'ontologia. HermiT è un reasoner (motore di inferenza) per ontologie basate su OWL, sviluppato in Java (necessario per eseguire questo progetto, vedere Requisiti funzionali). Quindi ci permette di eseguire ragionamenti logici su ontologie, verificandone la coerenza e deducendo nuove informazioni implicite a partire dalle asserzioni esplicite presenti nell'ontologia.

# Capitolo 2

## Ontologia

Una ontologia è una descrizione dei simboli usati dal sistema di informazione. Le ontologie sono in genere indipendenti da una particolare applicazione, basandosi su una comunità che concorda sul significato dei simboli. Un'ontologia consiste in:

- un vocabolario di classi e proprietà
- una struttura organizzativa
- un insieme di assiomi che restringono il significato del simbolo in modo da rispecchiare il significato inteso

Le ontologie moderne, come OWL, sono basate sulla logica descrittiva, usata per descrivere: individui, ovvero gli oggetti del mondo, classi, ovvero insiemi di individui, e proprietà, per descrivere le relazioni binarie tra individui ed altri individui o valori. La logica descrittiva distingue tra:

- terminological knowledge base o TBox che definisce il significato dei simboli; è definita nella fase di progettazione del sistema e cambia solo quando cambia il vocabolario
- assertional knowledge base o ABox definisce ciò che è vero in un dato istante temporale; contiene conoscenza che è specifica per il contesto ed è pertanto disponibile solo a run time

Per effettuare l'apprendimento del Knowledge Graph, sfruttiamo le triple ricavate dall'ontologia. Ogni tripla è espressa come  $\text{prop}(\text{soggetto}, \text{verbo}, \text{oggetto})$  ed indica che un individuo (soggetto) ha valore (oggetto) per una data proprietà (verbo). Il verbo della tripla è quindi una proprietà, il dominio è formato da quelle entità che possono comparire come soggetto per una particolare proprietà, il range è formato dalle entità o valori che possono comparire come oggetto per una particolare proprietà. Ogni individuo è denotato con un identificatore univoco.

## 2.1 Campi dell'ontologia

In particolare, per il dataset generato da Faker, gli individui (entità) sono:

- Person
- Course

mentre le relazioni sono:

- teaches, tra una persona ed un corso, per indicare i corsi che una persona insegna
- takes, tra una persona ed un corso, per indicare i corsi che una persona ha frequentato
- has\_age, tra una persona ed un valore numerico, indica l'età della persona
- has\_name, tra una persona ed un valore alfanumerico, indica il nome della persona
- course\_title, tra un corso ed un valore alfanumerico, indica il nome del corso

# Capitolo 3

## Funzionalità

### 3.1 Modello predittivo

Attraverso il file `dataset.csv` ottenuto dall'ontologia, inizia ad effettuare una serie di operazione di preprocessing e feature engineering, suddividere il dataset in insiemi di addestramento e test, standardizzare i dati, addestrare un modello di regressione logistica per la classificazione e, infine, restituire il modello addestrato insieme allo scaler usato e ad alcune metriche di valutazione.

- Divide il dataset in due parti: una per l'addestramento (`X_train`, `y_train`) e una per il test (`X_test`, `y_test`)
- Il parametro `test_size` indica la porzione di dati destinata al test set. Con un valore predefinito `test_size`, il 70% dei dati viene usato come test set e il restante 30% per l'addestramento.
- L'uso di `random_state` garantisce la riproducibilità della suddivisione.
- Viene istanziato un oggetto `StandardScaler` che standardizza le feature portandole a media 0 e varianza 1.
- `fit_transform` calcola la media e la deviazione standard per ogni feature sul training set e applica la trasformazione.
- **Trasformazione del test set:** Utilizza i parametri della media e deviazione standard calcolati sul training set per trasformare anche i dati del test set, evitando informazioni di leakage.
- Viene creato un modello di regressione logistica con un massimo di 1000 iterate (utile se il modello non converge rapidamente).
- `class_weight='balanced'` viene usato per bilanciare eventuali squilibri nelle classi.
- Il metodo `fit` addestra il modello utilizzando il training set standardizzato e le etichette corrispondenti.
- **Predizione:** Il modello addestrato viene utilizzato per predire le etichette del test set standardizzato (`y_test`)

- **Accuratezza:** `accuracy_score` misura la percentuale di predizioni corrette confrontando `y_pred` e `y_test`
- **Report di classificazione:** `classification_report` genera un report che include, per ogni classe, le metriche di Precision, Recall, F1-Score e Support (numero di occorrenze).  
L'argomento `zero_division=0` evita errori di divisione per 0, assegnando il valore 0 in quei casi.

## 3.2 Modello predittivo con Grid Search

Attraverso il file `dataset.csv` ottenuto dall'ontologia, permette non solo di addestrare un modello di regressione logistica, ma anche di trovare automaticamente combinazione di iperparametri che offre le migliori prestazioni, garantendo al contempo la corretta standardizzazione e gestione delle feature, oltre a considerare eventuali squilibri nel dataset.

- Divide il dataset in due parti: una per l'addestramento (`X_train`, `y_train`) e una per il test (`X_test`, `y_test`).
- Il parametro `test_size` indica la porzione di dati destinata al test set. Con un valore predefinito `test_size`, il 70% dei dati viene usato come test set e il restante 30% per l'addestramento.
- L'uso di `random_state` garantisce la riproducibilità della suddivisione.
- Viene definita una pipeline che contiene due passaggi:
  1. `StandardScaler`: Standardizza le feature (media 0 e varianza 1), operazione fondamentale per la regressione logistica.
  2. `LogisticRegression`: Il classificatore utilizzato, configurato con:
    - `random_state` per la riproducibilità.
    - `max_iter=1000` per garantire la convergenza (molte iterazioni se necessario).
    - `class_weight='balanced'` per gestire eventuali squilibri nelle classi, assegnando pesi maggiori alla classe meno rappresentata (nel nostro caso quasi sempre gli insegnanti).
- **Griglia degli iperparametri:**
  - `clf_`: i parametri vanno applicati all'oggetto `clf` all'interno della pipeline.
  - `C`: il parametro di regolarizzazione, esplorato in una scala logaritmica da  $10^{-5}$  a  $10^5$  con 11 valori distinti.
  - `penalty`: Il tipo di penalizzazione, che può essere `l1` o `l2`.
  - `solver`: Gli algoritmi utilizzati per l'ottimizzazione, differenziati a seconda del tipo di penalizzazione:
    - \* Per `l1`, vengono testati `liblinear` e `saga`.
    - \* Per `l2`, vengono testati `liblinear`, `saga`, `newton-cg` e `lbfgs`.
- **GridSearchCV**: Effettua una ricerca esaustiva nella griglia degli iperparametri definita:
  - `pipeline`: Il modello da ottimizzare (comprensivo di standardizzazione e regressione logistica).
  - `param_grid`: Griglia degli iperparametri da testare.

- `cv=5`: Viene utilizzata la cross-validation a 5-fold.
- `scoring='balanced_accuracy'`: La metrica scelta, utile per dataset sbilanciati, che calcola la media dell'accuratezza ottenuta per ciascuna classe.
- `n_jobs=-1`: Permette di eseguire i calcoli in parallelo su tutti i core disponibili.
- Viene eseguito il fitting della pipeline sul set di addestramento.
- Durante questo processo, per ogni combinazione degli iperparametri nella griglia, viene eseguita una cross-validation a 5 fold.
- Al termine, GridSearchCV seleziona il modello che ha ottenuto il punteggio migliore in base alla metrica `balanced_accuracy`.
- Viene estratto dalla Grid Search il miglior modello addestrato (la pipeline ottimizzata con i parametri migliori).
- **Predizione:** Il modello migliore viene usato per predire le etichette sul test set (`y_pred`)
- **Accuratezza:** `accuracy_score` misura la percentuale di predizioni corrette confrontando `y_pred` e `y_test`.
- **Report di classificazione:** `classification_report` genera un report che include, per ogni classe, le metriche di Precision, Recall, F1-Score e Support (numero di occorrenze).

### 3.2.1 Cross-validation a 5 fold

La **cross-validation a 5-fold** è una tecnica di validazione incrociata utilizzata per valutare la capacità di generalizzazione di un modello.

1. Il dataset viene diviso in 5 parti (fold) di dimensioni approssimativamente uguali.
2. Per ciascuno dei 5 fold, il procedimento è il seguente:
  - **Fold di validazione:** Viene selezionato uno dei 5 fold come set di test.
  - **Fold di addestramento:** Gli altri 4 fold vengono combinati per formare il set di addestramento.
  - Il modello viene addestrato sui 4 fold di addestramento e successivamente testato sul fold di validazione.
3. L'operazione viene ripetuta 5 volte, ognuna utilizzando un fold diverso come set di validazione. Ogni campione del dataset viene usato una volta per la validazione e 4 volte per l'addestramento.
4. Si calcola una metrica (es: accuratezza, F1-Score, ...) per ogni iterazione e alla fine si fa la media dei risultati ottenuti in tutti i 5 fold. Questo valore medio fornisce una stima più robusta e affidabile delle prestazioni del modello su dati non visti.

La cross-validation a 5 fold permette di ottenere una stima affidabile e robusta delle prestazioni di un modello, sfruttando l'intero dataset in modo efficiente e riducendo il rischio di errori dovuti a una particolare suddivisione dei dati.

### 3.3 Confronto tra modello predittivo base e modello predittivo con Grid Search

La differenza maggiore tra il primo modello predittivo e quello che utilizza il Grid Search risiede nel processo di **ottimizzazione degli iperparametri** e nella metodologia di **validazione** adottata:

#### 1. Ottimizzazione degli iperparametri:

- Nel primo approccio, il modello di regressione logistica viene addestrato con una configurazione fissa (`max_iter`, `class_weight`, ...) senza cercare di ottimizzare ulteriormente le impostazioni. Non viene esplorata la possibilità di migliorare le performance modificando, il parametro di regolarizzazione  $C$ , il tipo di penalizzazione o il solver.
- Nel secondo approccio, viene utilizzata una **Grid Search** (`GridSearchCV`) per esplorare sistematicamente un insieme di combinazioni di iperparametri (es: differenti valori di  $C$ , tipi di penalizzazione 11 e 12, e vari solver come `liblinear`, `saga`, `newton-cg`, `lbfgs`). Questo permette di trovare una configurazione ottimale che massimizza la metrica di valutazione adattando il modello alle peculiarità del dataset.

#### 2. Metodologia di validazione:

- Nel primo approccio, viene effettuata una singola suddivisione del dataset in training e test set. Il modello viene addestrato sui dati di training e poi valutato una sola volta sui dati di test. Questo approccio fornisce una stima delle performance, ma è più sensibile a come avviene la divisione dei dati.
- La Grid Search utilizza una **cross-validation a 5 fold**.

## 3.4 Come funziona PyKEEN con TransE

Data un'ontologia, siamo in grado di:

- Gestire l'estrazione di triplette da un'ontologia.
- Addestrare un modello di embedding per Knowledge Graph (KG) - usando PyKEEN (libreria Python).
- Visualizzare gli embeddings mediante tecniche di riduzione della dimensionalità (PCA, t-SNE).

Prima di tutto, dobbiamo assicurarci che la nostra ontologia sia stata caricata. In tal caso, si prova ad accedere alla classe `Person` definita nell'ontologia. Per ogni persona

- Si usa un identificativo (head) - `person.name`.
- Se la persona ha dei corsi che segue e questo attributo è valorizzato, per ogni corso viene aggiunta la tripla

```
(nome_persona, "takes", nome_corso)
```

- Se la persona ha dei corsi che insegna e questo attributo è valorizzato, per ogni corso viene aggiunta la tripla

```
(nome_persona, "teaches", nome_corso)
```

- Le triplette successivamente vengono convertite in un array numPy, e si inizia a preparare il dataset per addestrare il modello di embedding (tramite la funzione `TriplesFactory.from_labeled_triples()`).
- Il dataset viene suddiviso in tre parti con proporzioni: 80% per l'addestramento, 10% per il test e 10% per la validazione.
- Utilizzando la funzione `pipeline` fornita da PyKeen, viene addestrato un modello di tipo **TransE** per 100 epoch. Il risultato contiene il modello addestrato e altre informazioni.

Il modello viene ottimizzato iterativamente sull'intero dataset di triplette per 100 volte.

TransE è un algoritmo di embedding per knowledge graph che rappresenta le entità e le relazioni in uno spazio vettoriale continuo.

L'idea centrale di TransE è che, per una tripla  $(h, r, t)$  (dove  $h, r, t$ ), dove:

- $h$  è l'entità di partenza.
- $r$  la relazione.
- $t$  è l'entità di destinazione.

Il modello cerca di far sì che il vettore dell'entità  $h$  sommato al vettore della relazione  $r$  sia vicino (in termini di distanza, come la Distanza Euclidea o L1) al vettore entità  $t$ . Quindi, si cerca di approssimare

$$h + r \approx t$$

Un'epoca (**epoch**) corrisponde a un ciclo completo attraverso l'intero dataset di triplette. Durante ogni epoca, il modello esamina ogni tripla disponibile e aggiorna i propri parametri per ridurre l'errore nel rappresentare correttamente la struttura del Knowledge Graph.

- Dal modello si recupera la rappresentazione delle entità, poi viene trasformato l'id della persona nel suo nome.
- Alla fine è possibile rappresentare il modello di embedding con due dimensionalità diverse:
  - **PCA 2D-3D** (Principal Component Analysis): Mira a ridurre la dimensionalità dei dati trasformandoli in un nuovo insieme di variabili non correlate chiamate **componenti principali**. Queste componenti sono scelte in modo tale da catturare la massima varianza possibile presente nei dati.

Viene calcolata **varianza** di ciascuna feature e le correlazioni tra di esse.

Attraverso una decomposizione si determinano gli **autovettori** che indicano le direzioni (o assi) lungo le quali la varianza è massima. Questi autovettori sono le componenti principali.

I dati vengono proiettati su un sottospazio definito dalle prime  $k$  componenti principali, dove  $k$  è il numero di dimensioni ridotte che si desidera ottenere (es: 2 per una visualizzazione 2D o 3 per una visualizzazione 3D).

La trasformazione lineare produce nuove componenti principali che sono combinazioni lineari delle feature originali e che spiegano la maggior parte della varianza totale dei dati. La scelta del numero  $k$  di componenti da mantenere dipende dal compromesso tra semplificazione e perdita d'informazione: solitamente si sceglie un numero che spiega una percentuale elevata della varianza (es: 90-95%).

Un esempio lo possiamo trovare PCA2D o PCA3D.

- **t-SNE** (t-Distributed Stochastic Neighbor Embedding): Progettato per ridurre la dimensionalità dei dati in maniera non lineare, preservando in particolare le relazioni di vicinanza locali tra i punti.

Per ogni coppia di punti, t-SNE calcola una misura di similarità nel dataset originale, tipicamente trasformando le distanze in probabilità che rappresentano la vicinanza. Successivamente, t-SNE cerca di creare una mappa a bassa dimensionalità in cui le stesse relazioni di vicinanza siano mantenute quanto più possibile. Questo viene fatto minimizzando una funzione di costo che misura la differenza tra le distribuzioni di probabilità delle distanze nei due spazi.

Un parametro importante in t-SNE è la **perplessità**, che controlla il bilanciamento tra l'attenzione data alle strutture locali rispetto a quelle globali.

t-SNE genera una rappresentazione a bassa dimensionalità dove i punti che erano vicini nello spazio originale tendono a rimanere vicini, rendendo evidenti eventuali cluster o gruppi nei dati. Tuttavia, a causa della sua natura non lineare, le distanze globali (cioè tra cluster diversi) potrebbero non riflettere fedelmente le distanze reali.

Un esempio lo possiamo trovare qui.

# Capitolo 4

## Requisiti funzionali

### 4.1 Prerequisiti

- Python 3.6 o superiore.
- Le dipendenze elencate in `requirements.txt` (installabili con `pip install -r requirements.txt`).
- Java 21 o superiore.

#### 4.1.1 Installazione di Java

Per eseguire alcune parti del nostro progetto, è necessario avere installato il Java Development Kit (JDK). Di seguito sono riportate le istruzioni per installare Java sui principali sistemi operativi.

##### Windows

1. **Scarica il JDK:** Visita il sito ufficiale di Oracle a <https://www.oracle.com/java/technologies/javase-downloads.html> oppure utilizza una distribuzione open-source come **Adoptium/Temurin** all'indirizzo <https://adoptium.net/>.
2. **Installa il JDK:** Esegui il file di installazione scaricato e segui le istruzioni a schermo.
3. **Verifica l'installazione:** Apri il Prompt dei Comandi e digita:

```
java --version
```

Dovresti vedere un output che conferma la versione di Java installata.

##### Linux (Ubuntu)

1. **Aggiorna i repository:** Apri un terminale ed esegui:

```
sudo apt update
```

2. **Installa OpenJDK:** Per installare OpenJDK 11, esegui:

```
sudo apt install openjdk-11-jdk
```

3. **Verifica l'installazione:** Digita nel terminale:

```
java --version
```

Dovresti vedere l'output relativo alla versione di OpenJDK installata.

## macOS

1. **Utilizzo di Homebrew:** Se non hai già Homebrew installato, segui le istruzioni sul sito ufficiale (<https://brew.sh/>). Una volta installato, apri il Terminale ed esegui:

```
brew update  
brew install openjdk
```

2. **Configura Java:** Dopo l'installazione, potrebbe essere necessario aggiungere Java al PATH del sistema: se stai utilizzando la shell zsh

```
echo 'export PATH="/opt/homebrew/opt/openjdk/bin:$PATH"' >> ~/.zshrc
```

Invece per bash

```
echo 'export PATH="/opt/homebrew/opt/openjdk/bin:$PATH"' >> ~/.bashrc
```

3. **Verifica l'installazione:** Digita nel Terminale:

```
java --version
```

Dovresti visualizzare la versione di OpenJDK installata.

## 4.2 Struttura del Progetto

La struttura delle cartelle è organizzata in modo modulare per separare le responsabilità.  
La gerarchia delle cartelle è la seguente:

```
dataset_builder_ontology/
  data/
    ontology.owl      % Ontologia OWL popolata (output)
    dataset.csv       % Dataset generato (output)
  src/
    flask/
      templates/
        compare.html
        grid_search.html
        index.html
        dataset.html
        train.html
      app.py           % Interfaccia web con Flask
    owl/
      logger_config.py
      ontology_manager.py
    predictive_model/
      predictive_model.py
      grid_search_model.py
      compare_models.py
    cli.py            % Confronto tra modelli
    pykeen_leaner/
      learn_knowledge.py
  requirements.txt
  README.md
```

# Capitolo 5

## Interfacce Utente: CLI e Web App

### 5.1 Interfaccia a Riga di Comando (CLI)

Il file `cli.py` gestisce le operazioni come:

- Popolare l'ontologia.
- Estrarre il dataset.
- Addestrare modelli predittivi (base e con Grid Search).
- Addestrare modelli neurali per apprendere embedding (PyKEEN).
- Visualizzare gli embedding.

#### 5.1.1 Esempi di Esecuzione dal CLI

Per eseguire le operazioni via CLI, aprire il terminale nella root del progetto ed eseguire:

```
1 Dataset Builder Ontology - Interfaccia a riga di comando
2
3 positional arguments:
4 {populate,extract,train,grid_search,compare_base_grid, learn_graph}
5 Comandi disponibili
6 populate          Popola l'ontologia con dati generati
7 extract           Estrae il dataset dall'ontologia
8 train             Addestra il modello predittivo sul dataset
9 grid_search       Addestra il modello predittivo utilizzando
10                  Grid Search con cross-validation
11 compare_base_grid Confronta le prestazioni tra il modello base
12                  e quello con Grid Search
13 learn_graph      Addestra il modello utilizzando TransE
14 options:
15   -h, --help       show this help message and exit}

# Popola l'ontologia:
python src/cli.py populate
```

```
1 2025-02-08 00:00:00,000 - dataset_generator - INFO - Popolamento dell'  
    ontologia con dati casuali...  
2 2025-02-08 00:00:00,000 - dataset_generator - INFO - Ontologia non  
    trovata, creazione di una nuova ontologia  
3 2025-02-08 00:00:00,000 - dataset_generator - INFO - Caricamento dell'  
    ontologia da data\ontology.owl...  
4 2025-02-08 00:00:00,000 - dataset_generator - INFO - Ontologia caricata  
    correttamente.  
5 2025-02-08 00:00:00,000 - dataset_generator - INFO - Ontologia popolata e  
    salvata in data\ontology.owl.
```

```
# Estrae il dataset e lo salva in data/dataset.csv:  
python src/cli.py extract
```

```
1 * Owlready2 * Running Hermit...  
2 * Owlready2 * Hermit took 1.5451467037200928 seconds  
3 * Owlready * (NB: only changes on entities loaded in Python are shown,  
    other changes are done but not listed)  
4 2025-02-08 00:00:00,000 - dataset_generator - INFO - Ragionamento  
    completato con successo.  
5 2025-02-08 00:00:00,000 - dataset_generator - INFO - Estrazione delle  
    caratteristiche dagli individui...  
6 2025-02-08 00:00:00,000 - dataset_generator - INFO - Numero di persone  
    trovate: 1000  
7 2025-02-08 00:00:00,000 - dataset_generator - INFO - Estrazione dati  
    completata con successo.  
8 2025-02-08 00:00:00,000 - dataset_generator - INFO - Dataset salvato in  
    data\dataset.csv.
```

```
# Addestra il modello base:  
python src/cli.py train
```

```
1 Addestramento del modello predittivo...  
2 Accuratezza: 0.8057142857142857  
3      precision  recall  f1-score   support  
4  
5          0       0.99     0.81     0.89      690  
6          1       0.03     0.40     0.06       10  
7  
8      accuracy                           0.81      700  
9      macro avg       0.51     0.61     0.47      700  
10 weighted avg       0.98     0.81     0.88      700  
11  
12 Modello addestrato e valutato.
```

```

# Addestra il modello ottimizzato con Grid Search:
python src/cli.py grid_search

1 Addestramento del modello predittivo con Grid Search...
2 Accuracy: 0.7171
3 Classification Report:
4      precision    recall  f1-score   support
5
6          0       0.99      0.72      0.83      690
7          1       0.03      0.50      0.05       10
8
9      accuracy                           0.72      700
10     macro avg       0.51      0.61      0.44      700
11  weighted avg       0.98      0.72      0.82      700
12
13 Modello ottimizzato addestrato con successo!

```

```

# Confronta le prestazioni tra il modello base e quello ottimizzato:
python src/cli.py compare_base_grid

```

```

1 Confronto tra il modello base e il modello grid search...
2 --- Base Model ---
3 Accuracy sul test set: 0.8057
4 Classification Report:
5      precision    recall  f1-score   support
6
7          0       0.99      0.81      0.89      690
8          1       0.03      0.40      0.06       10
9
10     accuracy                           0.81      700
11     macro avg       0.51      0.61      0.47      700
12  weighted avg       0.98      0.81      0.88      700
13
14
15 --- GridSearch Model ---
16 Accuracy sul test set: 0.7171
17 Classification Report:
18      precision    recall  f1-score   support
19
20          0       0.99      0.72      0.83      690
21          1       0.03      0.50      0.05       10
22
23     accuracy                           0.72      700
24     macro avg       0.51      0.61      0.44      700
25  weighted avg       0.98      0.72      0.82      700
26
27 Confronto completato.

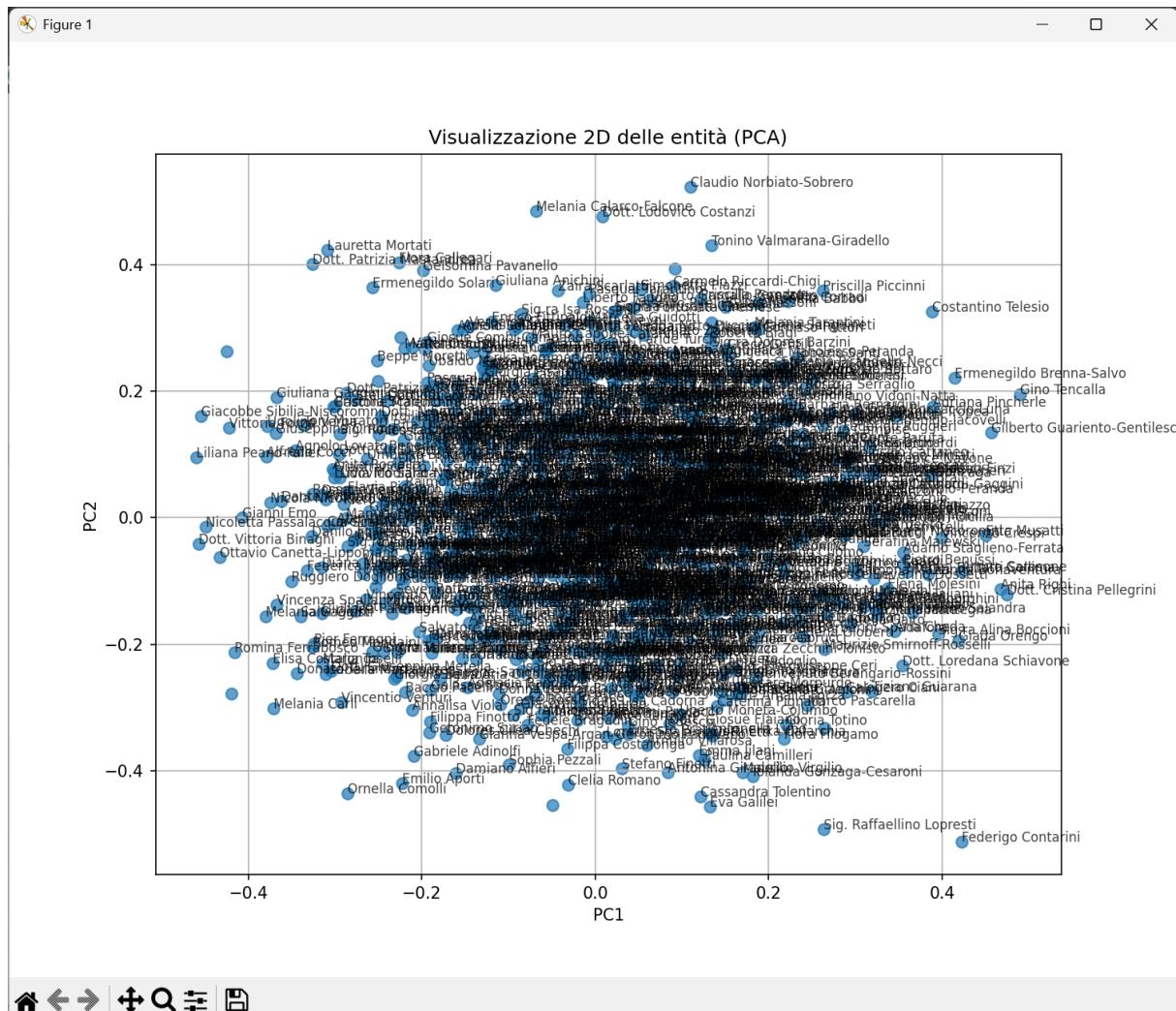
```

```
# Comando per addestrare il modello con TransE
python src/cli.py learn_graph
```

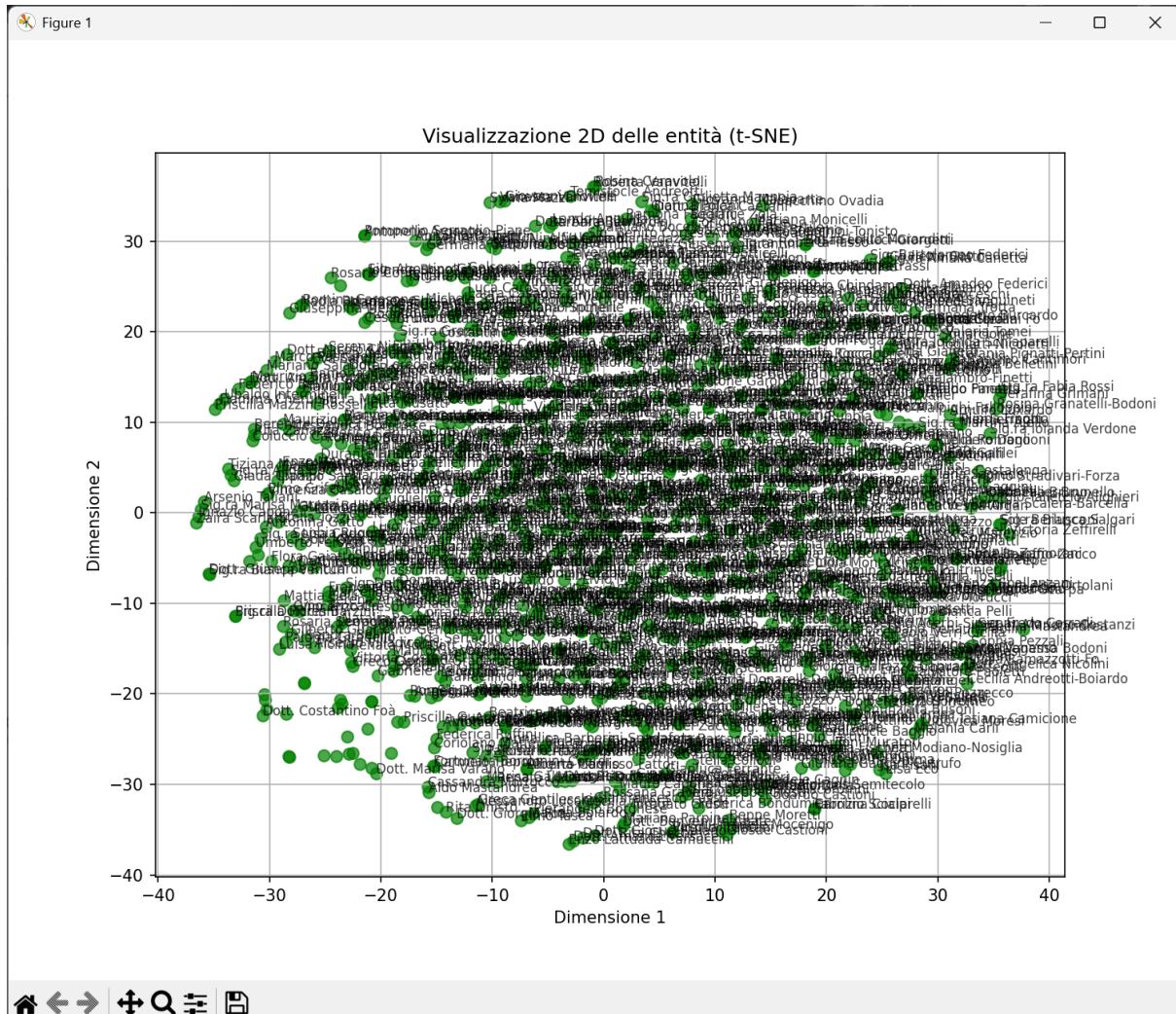
<sup>1</sup> Addestramento del modello su un dataset di triple...

dopodiché si apriranno in sequenza 3 finestre rappresentanti:

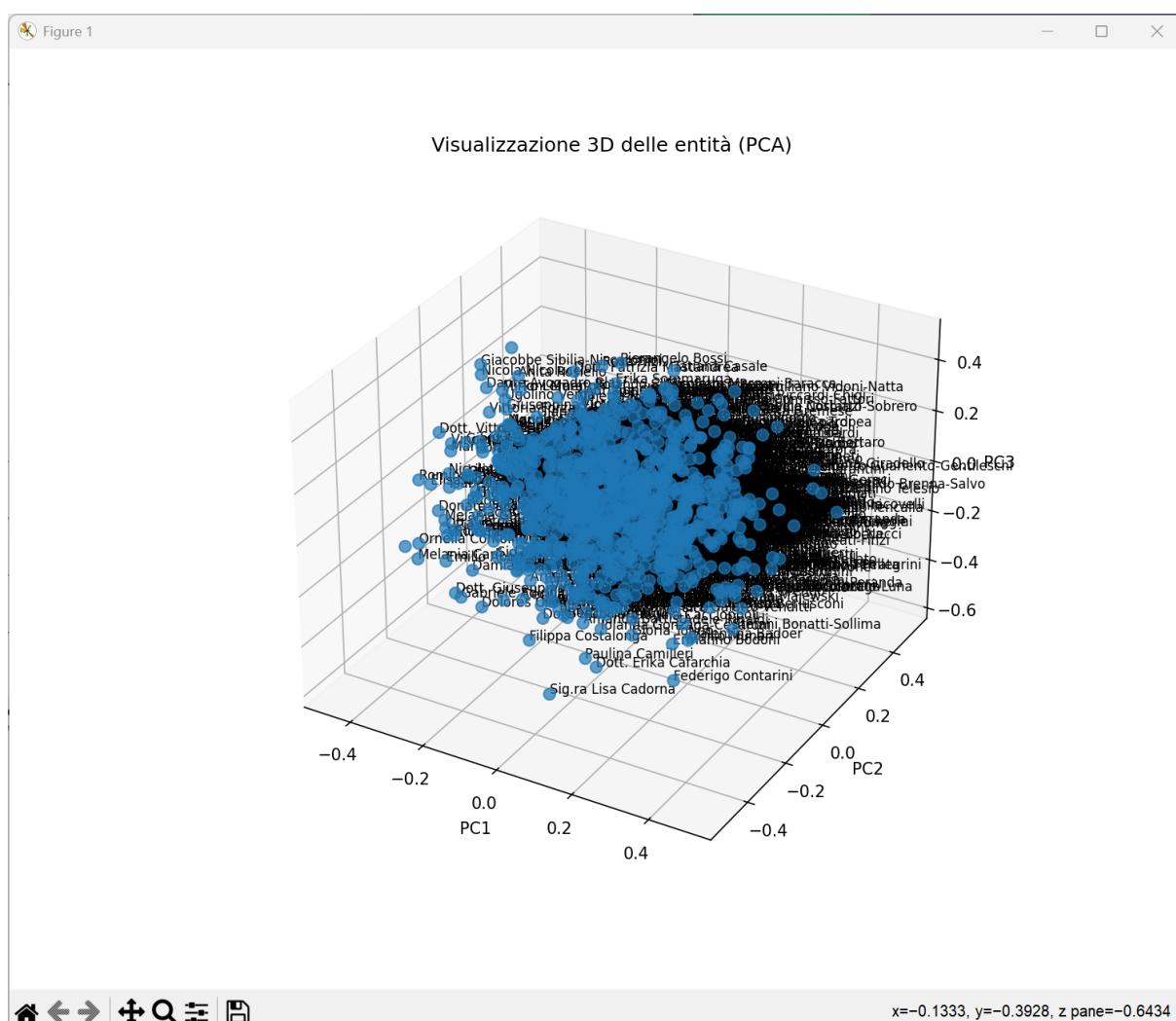
## PCA



## t-SNE



## PCA-3D



## 5.2 Interfaccia Web

L'applicazione web (definita in `app.py`) consente di eseguire le stesse operazioni tramite un'interfaccia grafica. Per avviare l'app:

```
python app.py
```

Successivamente, aprire il browser e navigare all'indirizzo:

```
http://127.0.0.1:4996/
```

Da qui è possibile interagire con l'applicazione per visualizzare il dataset estratto e i risultati dell'addestramento del modello.

### Ontology Dataset Builder

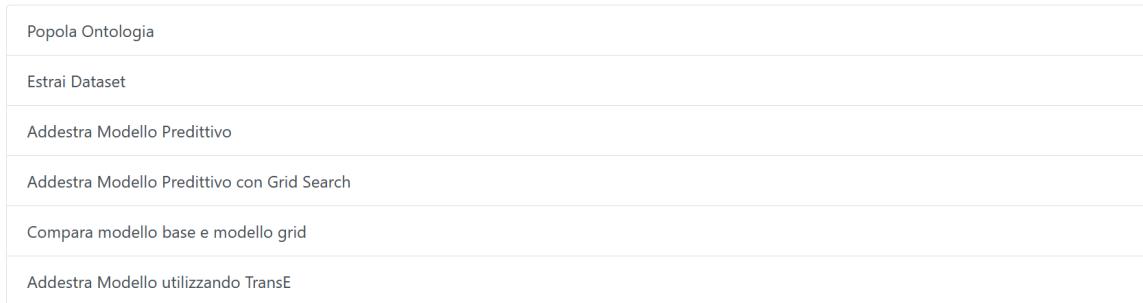
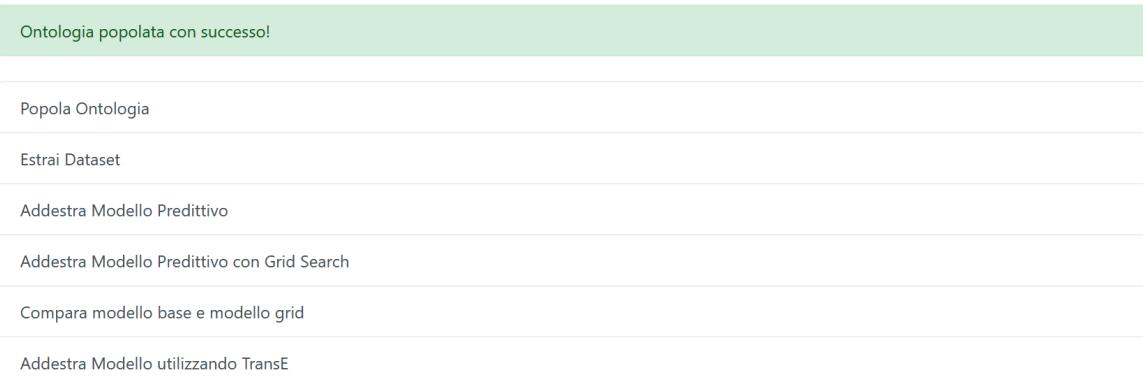


Figura 5.1: Schermata iniziale della web app Flask

1. **Popolare l'ontologia:** Cliccare il pulsante Popola Ontologia, aspettare qualche secondo, e avrai l'ontologia pronta

### Ontology Dataset Builder



2. **Estrai dataset:** Cliccare sul pulsante Estrai Dataset, dopo aver schiacciato Popola Ontologia (importante, altrimenti non funzionerà), aspettare qualche secondo, e avrai il risultato del dataset in base alla ontologia popolata precedentemente.

## Dataset Estratto

[Torna alla Home](#)

Dataset estratto e salvato con successo!

name	age	random_noise	random_noise1	random_noise2	random_noise3	random_category	courses_taken	courses_taught
Benedetto Solari-Jilani	78	54.464173	-24.429138	84.866424	57.318609	A	Corso 25, Corso 41	NaN
Aldo Cremonesi	56	84.659049	-18.966257	-97.107812	-3.590256	C	Corso 28	NaN
Carolina Vitturi	54	-5.452946	85.582300	86.512963	-26.796520	B	Corso 43	NaN
Cirillo Aulenti	23	73.315209	-55.143928	-93.505712	66.529677	C	Corso 30, Corso 46	NaN
Gian Carullo	64	-91.988814	20.604777	-16.177823	-51.933341	C	Corso 33, Corso 4	NaN
Aurora Travaglio-Lancisi	73	-38.478626	-77.287932	65.785873	-6.508622	A	Corso 18, Corso 7	NaN
Adele Vergerio	58	30.412310	-50.430832	-51.758709	-74.598602	B	Corso 42, Corso 38	NaN
Tiziana Zanichelli	26	70.389193	66.411974	76.599385	38.577417	B	Corso 11, Corso 37, Corso 31, Corso 47	NaN

**3. Addestra il modello predittivo:** Cliccando il pulsante Addestra Modello Predittivo, dopo aver popolato l'ontologia e aver estratto il dataset, avrai il risultato del training con il modello base.

## Risultati dell'Addestramento del Modello

Modello addestrato con successo!

Training Set Ratio (0-1)

0,7



[Applica modifiche](#)

### Risultati:

Accuracy: 0.6771

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.68	0.81	694
1	0.00	0.00	0.00	6
accuracy		0.68	0.68	700
macro avg	0.49	0.34	0.40	700
weighted avg	0.98	0.68	0.80	700

[Torna alla Home](#)

4. **Addestra il modello con Grid Search:** Cliccando il pulsante Addestra Modello Predittivo, dopo aver popolato l'ontologia e aver estratto il dataset, avrai il risultato del training con il modello Grid Search, che a differenza del modello base presenta qualche ottimizzazione (vedere sezione Modello Grid Search).

## Risultati dell'Addestramento del Modello

Modello addestrato con Grid Search!

Training Set Ratio (0-1)

0,7



[Applica modifiche](#)

### Risultati:

Accuracy: 0.9914

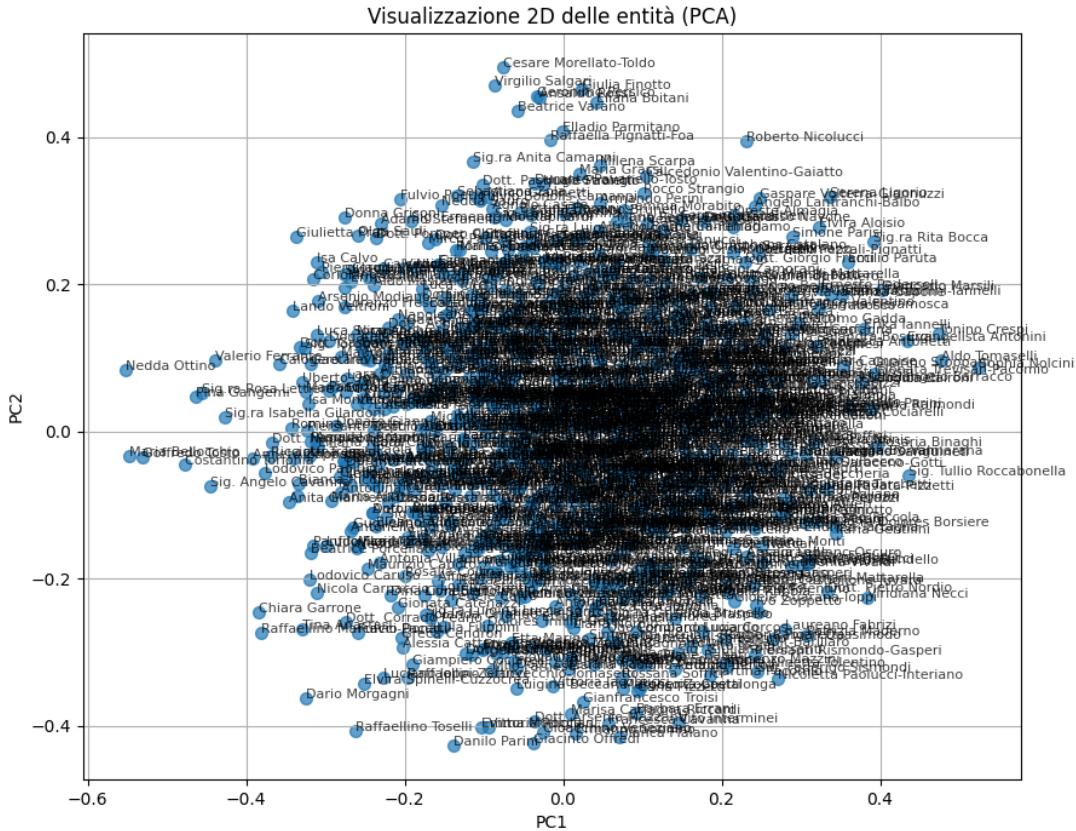
Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	694
1	0.00	0.00	0.00	6
accuracy		0.99	0.99	700
macro avg	0.50	0.50	0.50	700
weighted avg	0.98	0.99	0.99	700

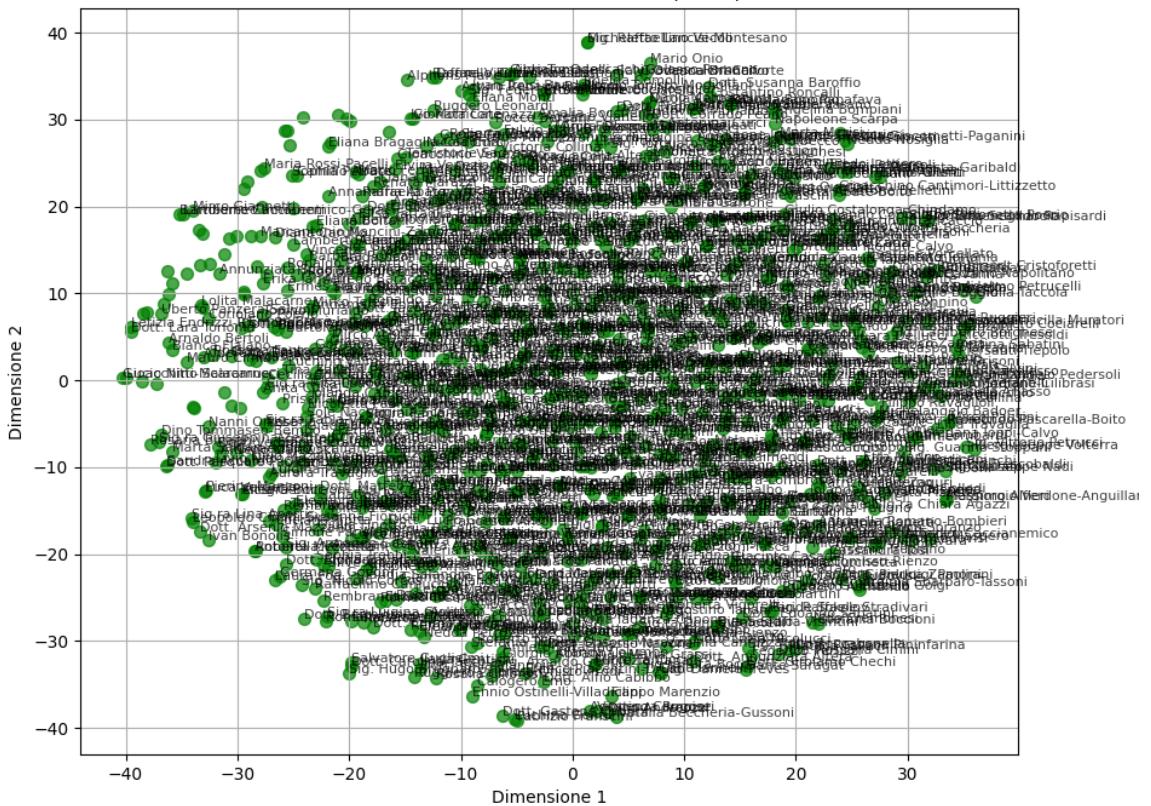
[Torna alla Home](#)

5. **Addestra Modello utilizzando TransE:** Cliccando il pulsante Addestra Modello utilizzando TransE, dopo aver popolato l'ontologia e aver estratto il dataset,

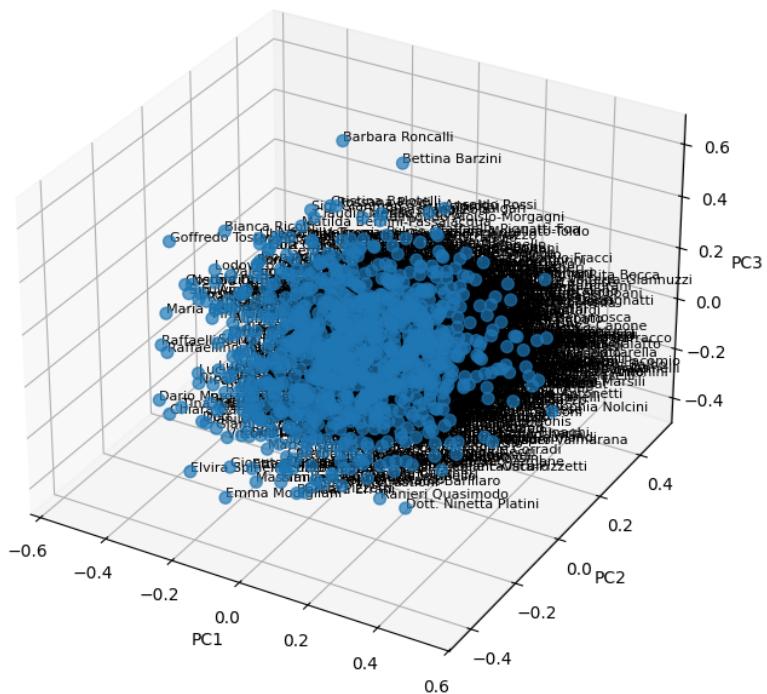
aspettando qualche minuto, avrai il risultato attraverso 3 grafici di dimensionalità diverse: PCA, t-SNE, PCA3D



Visualizzazione 2D delle entità (t-SNE)



## Visualizzazione 3D delle entità (PCA)



# Capitolo 6

## Conclusioni

Il progetto **Dataset Builder Ontology** rappresenta una piattaforma integrata per:

- La creazione e la gestione di ontologie.
- L'estrazione e il preprocessing di dataset da ontologie.
- L'addestramento di modelli predittivi tradizionali e neurali, con ottimizzazione automatica degli iperparametri.
- La generazione di embedding semantici (modelli PyKEEN) e la loro analisi tramite visualizzazione e clustering.
- L'interazione tramite CLI e una web app interattiva per configurazioni e monitoraggio dei processi.

Gli sviluppi futuri potrebbero includere:

- L'integrazione di ulteriori algoritmi di deep learning e interpretabilità dei modelli (ad es. SHAP o LIME).
- L'implementazione di dashboard interattive per monitorare le performance in tempo reale.
- L'espansione del knowledge graph con ulteriori dati e relazioni, migliorando così la qualità degli embedding.

# Bibliografia

- [1] David L. Poole and Alan K. Mackworth: *Artificial Intelligence: Foundations of Computational Agents, 3rd edition* Cambridge University Press 2023 (Ch.16) <https://artint.info/3e/html/ArtInt3e.Ch16.html>
- [2] Owlready2: A module for ontology-oriented programming in Python. <https://pypi.org/project/Owlready2/>
- [3] Scikit-learn Developers, Scikit-learn: Machine Learning in Python. <https://scikit-learn.org/>
- [4] Scikit-learn Developers, GridSearchCV Documentation. [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
- [5] Flask: A lightweight WSGI web application framework. <https://flask.palletsprojects.com/>
- [6] Galkin, M., et al. (2020). *PyKEEN 2.0: A Python Package for Training and Evaluating Knowledge Graph Embeddings*. <https://pykeen.readthedocs.io/>
- [7] Matplotlib: A 2D graphics environment. Available at: <https://matplotlib.org/>.
- [8] Array programming with NumPy. Available at: <https://numpy.org/>.
- [9] Bordes, A., Usunier, N., García-Durán, A., Weston, J., & Yakhnenko, O. (2013). *Translating Embeddings for Modeling Multi-relational Data*. In Advances in Neural Information Processing Systems (NIPS). <https://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf>