

Documentazione del Progetto:  
**Dataset Builder Ontology**

Vincenzo Buttari, Francesco Ciavarella

7 febbraio 2025

# Indice

<b>Prefazione</b>	<b>1</b>
<b>1 Struttura del Progetto</b>	<b>2</b>
<b>2 Gestione dell'Ontologia</b>	<b>3</b>
2.1 La Classe <code>OntologyManager</code> . . . . .	3
2.1.1 Esempio di Utilizzo . . . . .	3
<b>3 Modelli Predittivi Tradizionali e Ottimizzati</b>	<b>4</b>
3.1 Modello Base e Grid Search . . . . .	4
3.2 Modello Base . . . . .	4
3.2.1 Esempio di utilizzo . . . . .	4
3.3 Modello Grid Search . . . . .	5
3.3.1 Esempio di utilizzo . . . . .	5
3.4 Confronto delle Prestazioni . . . . .	5
3.4.1 Esempio di utilizzo . . . . .	6
<b>4 Utilizzo di Modelli Predefiniti con PyKEEN</b>	<b>7</b>
4.1 Visualizzazione e Clustering degli Embedding . . . . .	8
4.2 Riduzione della Dimensionalità con PCA e t-SNE . . . . .	8
<b>5 Interfacce Utente: CLI e Web App</b>	<b>10</b>
5.1 Interfaccia a Riga di Comando (CLI) . . . . .	10
5.1.1 Esempi di Esecuzione dal CLI . . . . .	10
5.2 Interfaccia Web . . . . .	11
<b>6 Istruzioni per l'Installazione</b>	<b>17</b>
6.1 Prerequisiti . . . . .	17
6.1.1 Installazione di Java . . . . .	17
6.2 Esecuzione . . . . .	18
<b>7 Conclusioni e Sviluppi Futuri</b>	<b>19</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**

7 febbraio 2025

# Prefazione

Questo documento descrive in dettaglio il progetto **Dataset Builder Ontology**. Il progetto nasce con l'obiettivo di:

- Popolare automaticamente un'ontologia OWL con dati realistici.
- Estrarre dati strutturati dall'ontologia per costruire un dataset.
- Addestrare modelli predittivi tradizionali (ad esempio, regressione logistica) e ottimizzati (tramite Grid Search) per attività di classificazione.
- Integrare modelli neurali per imparare rappresentazioni vettoriali (embeddings) degli oggetti dell'ontologia, sfruttando modelli predefiniti con PyKEEN.
- Visualizzare gli embedding tramite tecniche di riduzione della dimensionalità (PCA e t-SNE) e applicare algoritmi di clustering per identificare gruppi di entità simili.
- Fornire interfacce a riga di comando (CLI) e una web app con form interattivi per configurare la suddivisione del dataset (train/test).

Il documento illustra la struttura del progetto, i moduli principali, gli esempi di esecuzione e le possibili direzioni di sviluppo futuro.

# Capitolo 1

## 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 % Configurazione del logging  
      ontology_manager.py % Classe OntologyManager per gestione ontologia  
    predictive_model/  
      predictive_model.py % Modello predittivo base  
      grid_search_model.py % Ottimizzazione tramite Grid Search  
      compare_models.py % Confronto tra modelli  
    cli.py            % Interfaccia a riga di comando  
    pykeen_learner/  
      learn_knowledge.py % Implementazione di pyKEEN attraverso TransE  
  requirements.txt    % Dipendenze del progetto  
  README.md
```

# Capitolo 2

## Gestione dell'Ontologia

### 2.1 La Classe `OntologyManager`

Il modulo `ontology_manager.py` definisce la classe `OntologyManager`, che incapsula le seguenti operazioni:

- **`load()`**: Carica l'ontologia dal file specificato o, se il file non esiste, crea una nuova ontologia (file di default `ontology.owl`).
- **`populate()`**: Definisce classi (ad es. `Person` e `Course`), proprietà (ad es. `has_name`, `has_age`, `teaches`, `takes`, `course_title`, `course_description`) e popola l'ontologia con individui generati automaticamente.
- **`reason()`**: Esegue il ragionamento sull'ontologia tramite il ragionatore di `Owlready2`.
- **`extract_features()`**: Estrae le informazioni rilevanti degli individui della classe `Person` presenti nell'ontologia.

#### 2.1.1 Esempio di Utilizzo

```
1 from owl.ontology_manager import OntologyManager
2 import os
3
4 ontology_path = os.path.join("data", "ontology.owl")
5 onto = OntologyManager(ontology_file)
6
7 onto.populate() # Populate carica automaticamente la ontology
8 onto.extract_features() # Extract_features esegue il ragionamento sull'
   ontology
```

Listing 2.1: Utilizzo della classe `OntologyManager`

# Capitolo 3

## Modelli Predittivi Tradizionali e Ottimizzati

### 3.1 Modello Base e Grid Search

Il progetto addestra un modello di regressione logistica sul dataset estratto, utilizzando due approcci:

- **Modello Base:** Addestrato con parametri fissi.
- **Modello con Grid Search:** Utilizza `GridSearchCV` per ottimizzare iperparametri come  $C$  e il tipo di penalizzazione.

Le funzioni sono state aggiornate per utilizzare file di suddivisione (train/test) configurabili tramite un form della web app.

### 3.2 Modello Base

Il modulo `predictive_model.py` addestra un modello di regressione logica con parametri fissi utilizzando:

- La feature `age` e il numero di corsi seguiti.
- Un target `teacher` definito in base alla presenza di corsi insegnati.

#### 3.2.1 Esempio di utilizzo

```
1 from predictive_model.predictive_model import train_predictive_model
2
3 dataset_path = os.path.join("data", "dataset.csv")
4 base_model = train_predictive_model(dataset_path, random_state=42)
```

Listing 3.1: Utilizzo del modello predittivo base



### 3.3 Modello Grid Search

Il modulo `grid_search_model.py` implementa una procedura di Grid Search con 5-fold cross-validation per:

- Ottimizzare i parametri del modello (es: il parametro  $C$  e il tipo di penalizzazione 11 o 12).
- Trovare la configurazione ottimale in base all'accuracy media ottenuta durante la cross-validation.

Nel modello Grid search utilizziamo `np.logspace(-5,5,11)` per testare valori di  $C$  che vanno da  $10^{-5}$  a  $10^5$ . Questo permette di verificare se un'ampia variazione del grado di regolarizzazione influenza le prestazioni del modello.

- Per la **penalità 11**: testiamo i solver `liblinear` e `saga` (entrambi supportano 11)
- Per la **penalità 12**: testiamo `liblinear`, `saga`, `newton-cg` e `lbfgs`.

In questo modo, la Grid Search esplorerà combinazioni diverse per vedere se una configurazione ottimale riesce a gestire meglio il rumore e lo sbilanciamento.

Utilizziamo anche la metrica `balanced_accuracy` poiché è più adatta per problemi con classi sbilanciate, in quanto tiene conto del recall per entrambe le classi.

#### 3.3.1 Esempio di utilizzo

```
1 from predictive_model.grid_search import train_with_grid_search
2
3 dataset_path = os.path.join("data", "dataset.csv")
4 best_model = train_with_grid_search(dataset_path, random_state=42)
```

Listing 3.2: Utilizzo del modello predittivo con Grid Search

Per entrambi i modelli usiamo il parametro `class_weight = balanced` per assegnare pesi maggiori alla classe minoritaria.

### 3.4 Confronto delle Prestazioni

Il modulo `compare_models.py` confronta le prestazioni tra il modello base e quello ottimizzato:

- Suddivide il dataset in training e test set.
- Calcola l'accuracy e il classification report per ciascun modello.
- Stampa a video i risultati per facilitarne il confronto.

### 3.4.1 Esempio di utilizzo

```
1 from predictive_model.compare_models import compare_models
2
3 dataset_path = os.path.join("data", "dataset.csv")
4 compare_models(dataset_path)
```

Listing 3.3: Comparare i due modelli: base e grid search

## Capitolo 4

# Utilizzo di Modelli Predefiniti con PyKEEN

Utilizziamo PyKEEN per addestrare modelli di embedding sul knowledge graph. Le triple vengono caricate direttamente come lista di triple, poi suddivise in training, testing e validation tramite il metodo `split`. Esempio:

```
1 import numpy as np
2 from pykeen.pipeline import pipeline
3 from pykeen.triples import TriplesFactory
4 def extract_triples(self):
5     onto.load()
6     self.triples = []
7     try:
8         Person = onto.ontology.Person
9     except AttributeError:
10         raise ValueError("La classe 'Person' non è definita nell'
11             ontologia.")
12
13     for person in Person.instances():
14         head = person.name
15         if hasattr(person, "takes") and person.takes:
16             for course in person.takes:
17                 self.triples.append((head, "takes", course.name))
18         if hasattr(person, "teaches") and person.teaches:
19             for course in person.teaches:
20                 self.triples.append((head, "teaches", course.name))
21
22 # Converti la lista di triple in un array NumPy
23 def train_model(self):
24     self.extract_triples()
25     triples_array = np.array(self.triples)
26
27     tf = TriplesFactory.from_labeled_triples(triples_array)
28     print(tf)
29
30     tf_train, tf_test, tf_valid = tf.split([0.8, 0.1, 0.1])
31
32 result = pipeline(
33     training=tf_train,
34     testing=tf_test,
```

```

34     validation=tf_valid,
35     model="TransE", # ad es., usa il modello TransE
36     training_kwargs=dict(num_epochs=100),
37 )

```

Listing 4.1: Utilizzo di PyKEEN per il training degli embedding

## 4.1 Visualizzazione e Clustering degli Embedding

Una volta ottenuti gli embedding da PyKEEN, si possono esplorare in 2D/3D e analizzare la struttura semantica:

## 4.2 Riduzione della Dimensionalità con PCA e t-SNE

Utilizziamo PCA e t-SNE per ridurre la dimensionalità e visualizzare i dati:

```

1  import matplotlib.pyplot as plt
2  from sklearn.decomposition import PCA
3  from sklearn.manifold import TSNE
4
5  entity_embedding = result.model.entity_representations[0]
6  embeddings_tensor = entity_embedding()
7  self.embeddings = embeddings_tensor.detach().cpu().numpy()
8
9  entity_to_id = tf.entity_to_id
10 id_to_entity = {v: k for k, v in entity_to_id.items()}
11 self.entity_labels = [id_to_entity[i] for i in range(len(id_to_entity))]
12
13 self.pca()
14 self.tsne2D()
15 self.pca3D()
16
17 # # --- A) Usando PCA per una visualizzazione 2D
18 def pca(self):
19     pca = PCA(n_components=2)
20     embeddings_2d = pca.fit_transform(self.embeddings)
21
22     plt.figure(figsize=(10, 8))
23     plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], s=50, alpha
24                 =0.7)
25     for i, label in enumerate(self.entity_labels):
26         plt.annotate(self.find_name(label), (embeddings_2d[i, 0],
27         embeddings_2d[i, 1]), fontsize=8, alpha=0.75)
28     plt.title("Visualizzazione 2D delle entità (PCA)")
29     plt.xlabel("PC1")
30     plt.ylabel("PC2")
31     plt.grid(True)
32     plt.show()
33
34 # --- B) Usando t-SNE per una visualizzazione 2D (puo' evidenziare
35     strutture non lineari)
36 def tsne2D(self):
37     tsne_2d = TSNE(n_components=2, random_state=42, init="random")

```

```

35 embeddings_tsne_2d = tsne_2d.fit_transform(self.embeddings)
36
37 plt.figure(figsize=(10, 8))
38 plt.scatter(embeddings_tsne_2d[:, 0], embeddings_tsne_2d[:, 1], s
39             =50, alpha=0.7, c='green')
39 for i, label in enumerate(self.entity_labels):
40     plt.annotate(self.find_name(label), (embeddings_tsne_2d[i, 0],
41         embeddings_tsne_2d[i, 1]), fontsize=8, alpha=0.75)
41 plt.title("Visualizzazione 2D delle entità (t-SNE)")
42 plt.xlabel("Dimensione 1")
43 plt.ylabel("Dimensione 2")
44 plt.grid(True)
45 plt.show()
46
47 # --- C) Visualizzazione 3D con PCA
48 def pca3D(self):
49     pca_3d = PCA(n_components=3)
50     embeddings_3d = pca_3d.fit_transform(self.embeddings)
51
52     fig = plt.figure(figsize=(10, 8))
53     ax = fig.add_subplot(111, projection='3d')
54     ax.scatter(embeddings_3d[:, 0], embeddings_3d[:, 1], embeddings_3d[
55         :, 2], s=50, alpha=0.7)
55     for i, label in enumerate(self.entity_labels):
56         ax.text(embeddings_3d[i, 0], embeddings_3d[i, 1], embeddings_3d[
57             i, 2], self.find_name(label), size=8, zorder=1, color='k')
57     ax.set_title("Visualizzazione 3D delle entità (PCA)")
58     ax.set_xlabel("PC1")
59     ax.set_ylabel("PC2")
60     ax.set_zlabel("PC3")
61     plt.show()

```

Listing 4.2: Visualizzazione con PCA e t-SNE

# 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:

```
# Popola l'ontologia:
```

```
python src/cli.py populate
```

```
# Estrae il dataset e lo salva in data/dataset.csv:
```

```
python src/cli.py extract
```

```
# Addestra il modello base:
```

```
python src/cli.py train
```

```
# Addestra il modello ottimizzato con Grid Search:
```

```
python src/cli.py grid_search
```

```
# Confronta le prestazioni tra il modello base e quello ottimizzato:
```

```
python src/cli.py compare
```

```
# Comando per addestrare il modello con TransE
```

```
python src/cli.py learn_graph
```

## 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

Popola Ontologia
Estrai Dataset
Addestra Modello Predittivo
Addestra Modello Predittivo con Grid Search
Compara modello base e modello grid
Addestra Modello utilizzando TransE

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

Ontologia popolata con successo!
Popola Ontologia
Estrai Dataset
Addestra Modello Predittivo
Addestra Modello Predittivo con Grid Search
Compara modello base e modello grid
Addestra Modello utilizzando TransE

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	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

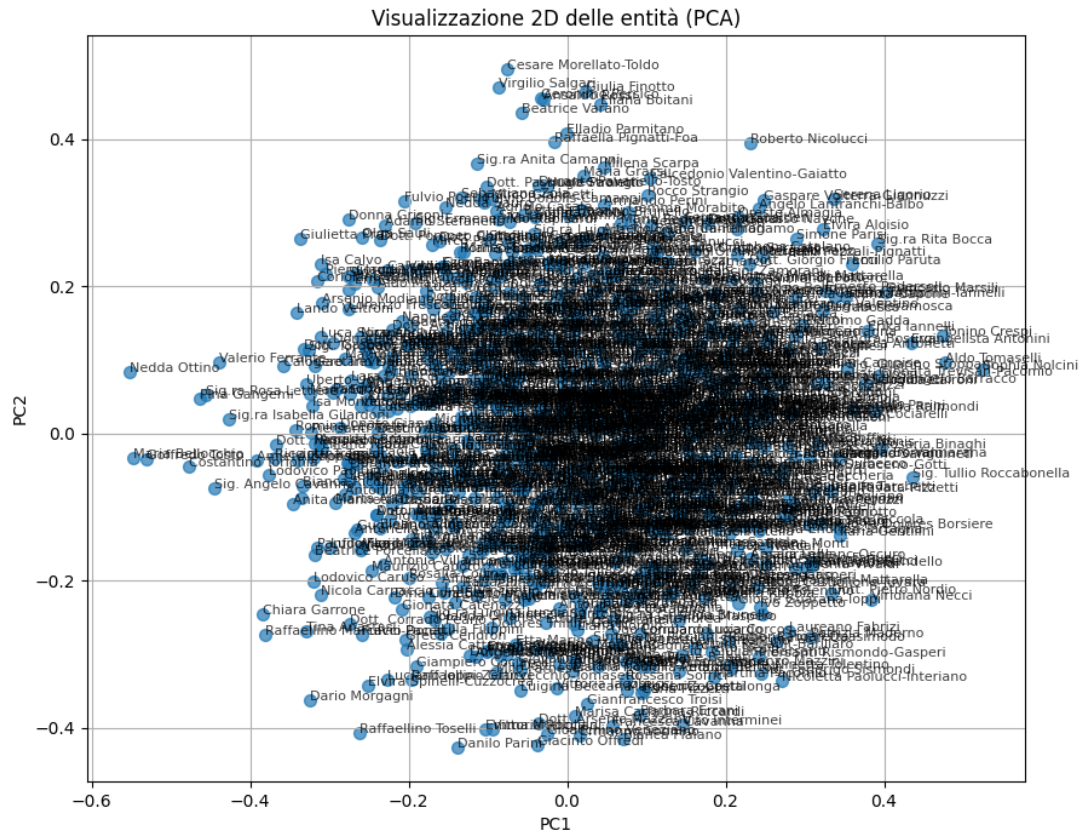
Classification Report:

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

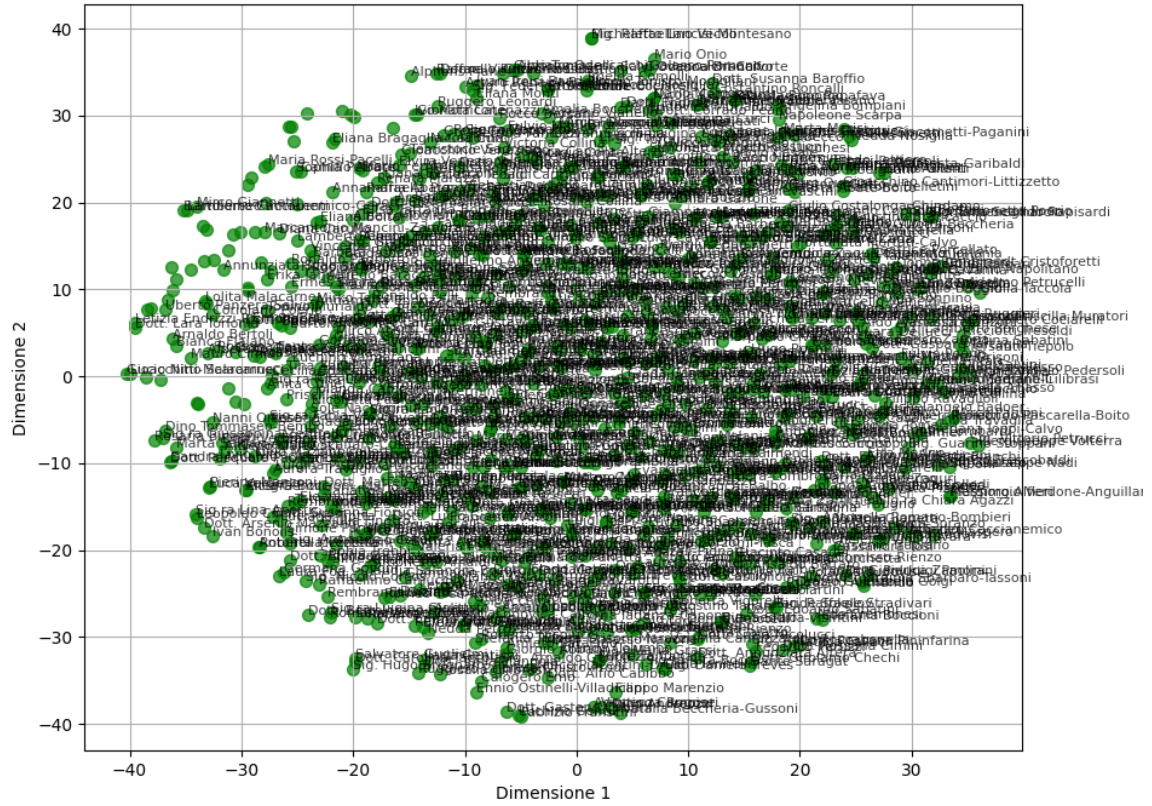
Torna alla Home

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

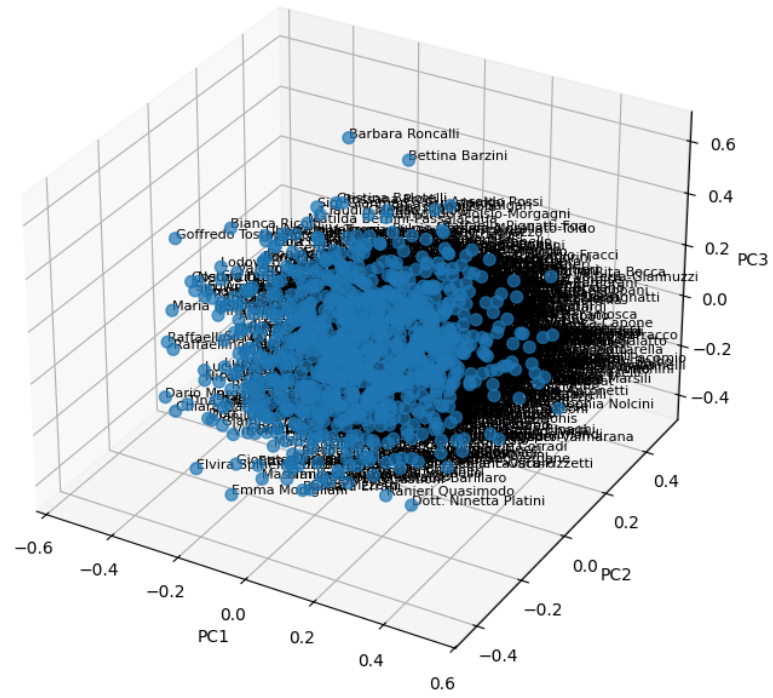
aspettando qualche minuto, avrai il risultato attraverso 3 grafici di dimensionalità diverse: PCA, t-SNE, PCA3D



### Visualizzazione 2D delle entità (t-SNE)



Visualizzazione 3D delle entità (PCA)



# Capitolo 6

## Istruzioni per l'Installazione

### 6.1 Prerequisiti

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

#### 6.1.1 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.

## 6.2 Esecuzione

- Per eseguire il progetto via CLI, utilizzare i comandi indicati nella sezione "Esempi di Esecuzione dal CLI".
- Per avviare l'interfaccia web, eseguire `python app.py` e accedere all'indirizzo `http://127.0.0.1:4996/`.

# Capitolo 7

## Conclusioni e Sviluppi Futuri

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] Owlready2: A module for ontology-oriented programming in Python. <https://pypi.org/project/Owlready2/>
- [2] Scikit-learn Developers, Scikit-learn: Machine Learning in Python. <https://scikit-learn.org/>
- [3] 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)
- [4] Flask: A lightweight WSGI web application framework. <https://flask.palletsprojects.com/>
- [5] PyKEEN: A Python library for learning and evaluating knowledge graph embeddings. <https://pykeen.readthedocs.io/>
- [6] Matplotlib: A 2D graphics environment. Available at: <https://matplotlib.org/>.
- [7] Array programming with NumPy. Available at: <https://numpy.org/>.