Report Project Assignment

Tecnologie Semantiche

UNIVERSITÀ DEGLI STUDI
DI SALERNO

Prof.ssa:

- **Sabrina Senatore**

Membri del gruppo:

Cesarano Mario	**0622701166**	**m.cesarano22@studenti.unisa.it**
D'Amore Grazia	**0622701462**	**g.damore15@studenti.unisa.it**

Studio del problema

Analisi dei dati

Si dispone di un file .csv contenente una serie di triple nel formato “subject,relation,object”; si assume che le varie triple appartengano allo stesso dominio, in modo tale da poter costruire nuove triple che possano essere successivamente classificate per la loro veridicità.

Analisi del contesto

A partire da un set di triple, si è addestrata una rete neurale, che puntasse a verificare la probabilità che nuove triple potessero essere vere. Nel caso in cui la probabilità calcolata superasse una certa soglia, la tripla corrispondente, verrebbe inserita all’interno di un grafo RDF.

Architettura del Sistema

Componenti del Sistema

AmpliGraph

AmpliGraph è una suite di modelli neurali per il machine learning utilizzati per l'apprendimento relazionale, un ramo del machine learning che sfrutta l'apprendimento supervisionato sui knowledge graph.

AmpliGraph può essere utilizzato principalmente per:

- inferire nuova conoscenza da un knowledge graph esistente.
- completare vasti knowledge graphs con statement mancanti.
- generare knowledge graph embeddings indipendenti.
- sviluppare e valutare un nuovo modello relazionale.

Nel nostro caso, AmpliGraph è stato utilizzato per addestrare un knowledge graph embeddings model e inferire nuova conoscenza dal grafo creato. AmpliGraph include le seguenti categorie:

- Datasets: funzioni utili a caricare i datasets (knowledge graphs).
- Models: modelli di knowledge graph embeddings. In particolare, AmpliGraph contiene TransE, DistMult, ComplEx, HolE, ConvE, ConvKB.
- Evaluation: protocolli di metriche e valutazioni per verificare il potere predittivo del modelli.
- Discovery: API alto-livello utili alla scoperta di nuova conoscenza.

Google COLAB

Google Colab è uno strumento gratuito presente nella suite Google che consente di scrivere codice python direttamente dal proprio browser. Una piattaforma online che offre un servizio di cloud hosting per notebook Jupyter dove creare ricchi documenti che contengono righe di codice, grafici, testi, link e molto altro. Il notebook Jupiter verrà poi eseguito su macchine virtuali di server Google. Ciò consente di svincolarsi dalla parte hardware e di concentrarci solamente sul codice Python e sui contenuti che si vuole integrare nel notebook.

Visual Studio

Visual Studio è un ambiente di sviluppo integrato sviluppato da Microsoft. Visual Studio è multilinguaggio e attualmente supporta la creazione di progetti per varie piattaforme, tra cui anche Mobile e Console. È possibile creare ed utilizzare estensioni e componenti aggiuntivi.

Neo4j

Neo4j è un software per basi di dati a grafo open source sviluppato interamente in Java. È un database totalmente transazionale, che viene integrato nelle applicazioni permettendone il funzionamento stand alone e memorizza tutti i dati in una cartella.
I dati non vengono memorizzati come astrazione di un grafo che si appoggia su di una ulteriore tecnologia, ma vengono memorizzati proprio come l'utente li definisce. Oltre ad un grafo, Neo4j fornisce anche la possibilità di effettuare le transazioni ACID e supporto ai cluster.

Neosemantics Plugin

Neosemantics (n10s) è un plugin che consente l'uso di RDF e dei suoi linguaggi associati (OWL, RDFS, SKOS e altri) in Neo4j.
Funzionalità incluse:

- Import/export di RDF e RDF* in più formati (Turtle, N-Triples, JSON-LD, RDF/XML, TriG and N-Quads, Turtle*, TriG*)
- Model Mapping su import/export
- Import ed export di Ontologie/Tassonomie in linguaggi differenti (OWL, SKOS, RDFS)
- Validazione dei grafi attraverso i vincoli SHACL
- Inferenza di base

Nel nostro caso, il plugin è stato usato per l'import di triple RDF nel formato N-Triples.

I/O dei Componenti

Ampligraph

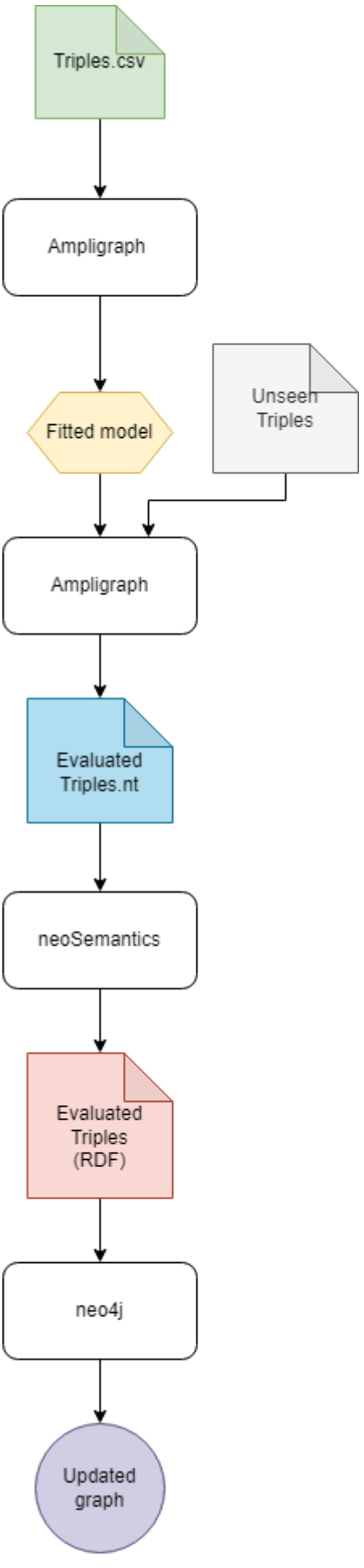
Questo tool è stato utilizzato principalmente per poter addestrare un modello di classificatore neurale sulle triple fornite in modo da poter inferire nuove triple. La prima fase prevede che l'input di questo modulo sia un file csv UTF-8 delimitato da virgole contenente l'insieme di triple sulla quale si vuole addestrare il modello, l'output sia il modello addestrato. La seconda fase utilizza come input il modello addestrato e una serie di triple esterne al set di training, l'output consiste in un DataFrame che contiene le triple e le metriche di valutazione (rank, score, probabilità). Le triple con le migliori metriche di valutazioni vengono salvate in un file in formato .nt (nTriples).

Neosemantics

Questo strumento è impiegato per inserire le triple estratte al passo precedente in un grafo RDF. Per cui l'input è il file .nt generato in precedenza, mentre l'output è il formato RDF delle triple stesse.

Neo4j

Questo tool è stato utilizzato per lo storing dei dati e la visualizzazione. In particolar modo, si è scelto di inserire le triple in formato RDF tramite il plugin neosemantics. Per cui, l'input consiste nelle triple in formato RDF di neosemantic e l'output è il grafo aggiornato.



Implementazione e Interfaccia

Soluzione proposta

Ampligraph/Colab

La soluzione al problema descritto in precedenza prevede una serie di passi da effettuare mediante i tool indicati. Inizialmente, sono state analizzate le varie caratteristiche di ognuna delle tecnologie utilizzabili; si è partiti facendo riferimento ad un file .csv fornitoci, contenente al suo interno un set di dati molto ampio, sotto forma di triple RDF (subject,relation,object). Osservando il file, è stato dedotto che si trattasse di asserzioni riguardanti la saga di Harry Potter. In seguito, si è utilizzato Google Colaboratory per poter processare le triple contenute nel file .csv. In particolare, ad ogni elemento della tripla è stato aggiunto il prefisso "http://dbpedia.org/resource/" in modo da ottenere degli URI validi che riportino alla pagina dbpedia relativa al soggetto o oggetto rappresentato dallo statement. Per lo stesso motivo, per ogni elemento eventuali spazi bianchi sono stati sostituiti da con un underscore (_).

```
import csv

df = pd.read_csv('Triple.csv', names=["subject","relation","object"])

dbpedia = "http://dbpedia.org/resource/"

for index,row in df.iterrows():
  s = dbpedia + row['subject'].replace(" ","_")
  p = dbpedia + row['relation'].replace(" ","_")
  o = dbpedia + row['object'].replace(" ","_")

  with open('HP_uri.csv', 'a+',newline='') as f:
      writer = csv.writer(f)
      writer.writerow([s,p,o])
      f.close()
```

Successivamente, è stato utilizzato Google Colaboratory per l'implementazione del codice riguardante il caricamento del set di dati e lo split dei dati per la fase di train e test.

```python
from ampligraph.evaluation import train_test_split_no_unseen

num_test = int(len(X) * (20 / 100))

data = {}
data['train'], data['test'] = train_test_split_no_unseen(X, test_size=num_test, seed=0, allow_duplication=False)

print('Train set size: ', data['train'].shape)
print('Test set size: ', data['test'].shape)
```

```
Train set size:  (861, 3)
Test set size:  (215, 3)
```

Una volta effettuato lo split, si è proceduti con l'addestramento del modello. La fase di addestramento è stata affrontata con un approccio prevalentemente empirico provando diversi modelli e diversi parametri sugli stessi dati e valutando quale di questi restituisse le metriche migliori sugli stessi campioni di test. La metrica principale è stata la MRR (Mean Reciprocal Rank)

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} = \frac{1}{rank_{(s,p,o)i}}$$ con (s,p,o) set di triple in Q

MRR considera il reciproco degli elementi di un vettore di ranking in modo da mantenere una maggiore robustezza agli outliers. Insieme ad MRR, è stata utilizzata la metrica Hits@n la quale fornisce il numero di elementi che sono stati classificati con un rank maggiore o uguale ad n. Nel nostro specifico caso sono stati considerati valori per n = 10,3,1. La scelta del modello è stata effettuata considerando i risultati sul dataset fornito da Ampligraph: 'https://ampligraph.s3-eu-west-1.amazonaws.com/datasets/GoT.csv'. Questo ha riportato il risultato migliore per il modello HolE.

```python
from ampligraph.latent_features import HolE

#optimizer
#adam     MRR = 0.44

#loss
#pairwise         MRR = 0.36

#regularizer
#'p':2  MRR = 0.44

#lr
#1e-4   MRR = 0.36
model = HolE(batches_count=100,
                seed=0,
                epochs=200,
                k=150,
                eta=5,
                optimizer='adam',
                optimizer_params={'lr':1e-2},
                loss='multiclass_nll',
                regularizer='LP',
                regularizer_params={'p':3, 'lambda':1e-5},
                verbose=True)
```

```python
from ampligraph.evaluation import evaluate_performance

filter_triples = np.concatenate((data['train'], data['test']))
ranks = evaluate_performance(data['test'],
                             model=model,
                             filter_triples=filter_triples,   # Corruption strategy filter defined above
                             use_default_protocol=True, # corrupt subj and obj separately while evaluating
                             verbose=True)
```

La stessa configurazione del modello ha riportato i seguenti risultati per il nostro dataset:

```
MRR: 0.73
Hits@10: 0.77
Hits@3: 0.75
Hits@1: 0.71
```

Infine, il modello addestrato è stato utilizzato per effettuare un'evaluation sulle triple unseen ottenendo 3 metriche principali : rank, score e probabilità.

```
from scipy.special import expit
probs = expit(scores)

output = pd.DataFrame(list(zip([' '.join(x) for x in X_unseen],
                        ranks_unseen,
                        np.squeeze(scores),
                        np.squeeze(probs))),
             columns=['statement', 'rank', 'score', 'prob']).sort_values("score")

output
```

	statement	rank	score	prob
	http://dbpedia.org/resource#Ron http://dbpedia...	1523	-0.638364	0.345616
	http://dbpedia.org/resource#Harry http://dbped...	1496	-0.517893	0.373345

Di ognuna delle triple unseen generate andava ovviamente valutata la veridicità, ed è stato deciso di utilizzare una soglia in relazione al rank fornito in output; le triple che possiedono un rank minore della soglia, vengono salvate all'interno di un file in formato .nt. Ciascuno statement con un rank al di sopra di una certa soglia viene processato in modo da rappresentare una tripla valida:

1. Eventuali spazi bianchi dello statement sono rimpiazzati da underscore (_)

```
for index, row in output.iterrows():
  if row['rank'] < th:
      statement = row['statement']

      namespace = "http://dbpedia.org/resource"

      url_subject,url_predicate,url_object = statement.split(" ")

      url_subject = url_subject.replace(" ","_")
      url_predicate = url_predicate.replace(" ","_")
      url_object = url_object.replace(" ","_")
```

2. Da ogni URI vengono estratti gli specifici elementi che rappresentano eliminando il prefisso. Per ogni soggetto/oggetto vengono specificate 3 triple:
- una tripla specifica il tipo che può essere “subject” o “object”

ES.

```
<http://dbpedia.org/resource#Hagrid>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/resource#subject> .
```

- una tripla specifica la proprietà “name”, cioè il nome del soggetto/oggetto in esame

ES.

```
<http://dbpedia.org/resource#Hagrid>
<http://dbpedia.org/resource#name>
"Hagrid" .
```

- una tripla specifica la proprietà “link”, cioè il link dbpedia corrispondente al soggetto/oggetto in esame.

```
subject = url_subject.replace("http://dbpedia.org/resource/" , "")
predicate = url_predicate.replace("http://dbpedia.org/resource/", "")
objectt = url_object.replace("http://dbpedia.org/resource/", "")

#Subject type
lines.append("<" + namespace + "#" + subject + ">" +
             " <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>" +
             " <" + namespace + "#subject> .")
#Subject name
lines.append("<" + namespace + "#" + subject + ">" +
             " <" + namespace + "#name>" +
             " \"" + subject + "\" .")
#Subject link
lines.append("<" + namespace + "#" + subject + ">" +
             " <" + namespace + "#link>" +
             " \"" + url_subject + "\" .")
```

3. Infine viene specificata la tripla che individua soggetto, predicato e oggetto

```
#Triple
s = "<" + namespace + "#" + subject + ">"
p = "<" + url_predicate + ">"
o = "<" + namespace + "#" + objectt + "> ."
lines.append(s + " " + p + " " + o)
```

Neosemantic/neo4j

Si è proceduti con l'utilizzo di neoSemantic per poter convertire le triple .nt in formato RDF e aggiungerle al grafo creato su neo4j. In particolare si è implementata un'interfaccia grafica che permetta di aggiungere delle triple al grafo neo4j in due modi:
1. caricando un insieme di triple tramite un file .nt. In particolare, viene utilizzato uno store chiamato "Neo4j" (o creato, se non esiste) e si specifica la configurazione del database alla quale ci si vuole connettere. In seguito, verrà chiesto di specificare il nome del file .nt dalla quale caricare le triple ed infine saranno caricate tramite la funzione "load" messa a disposizione da rdflib.

```
def load_triples():
    # create a neo4j backed Graph
    g = rdflib.Graph(store="Neo4j")

    # set the configuration to connect to your Neo4j DB
    theconfig = {'uri': "neo4j://127.0.0.1:7687",
                 'database': "neo4j",
                 'auth': {'user': "neo4j", 'pwd': "sas"}}

    g.open(theconfig, create = True)

    filename = askstring('Name', 'Qual è il nome del file di triple?')
    g.load(filename, format="nt")
    print("Done!")
```

2. selezionare una tripla scegliendo tra i possibili soggetti,predicati e oggetti. In particolare, si definisce il grafo e la configurazione del database a cui ci si vuole connettere in maniera analoga al caso precedente. In seguito, si specifica il namespace del database alla quale si sta facendo accesso. Da qui si definiscono singolarmente soggetto, predicato e oggetto. In questa specifica implementazione si riportano soggetti e oggetti con lo URI "http://dbpedia.org/resource#" seguito dal nome del soggetto/oggetto in esame. Sia per il soggetto che per l'oggetto sono specificate: la proprietà "name", la quale si aspetta un Literal che rappresenta il nome con cui è identificato; la proprietà "link", la quale si aspetta un Literal che rappresenta il link dbpedia corrispondente al soggetto/oggetto in esame. La scelta di non utilizzare direttamente quest'ultimo come URI permette di visualizzare meglio la differenza tra i nodi subject e i nodi object su neo4j. Infine, la tripla è

aggiunta tramite il metodo “add” messo a disposizione da rdflib che richiede di specificare il soggetto, il predicato e l’oggetto.

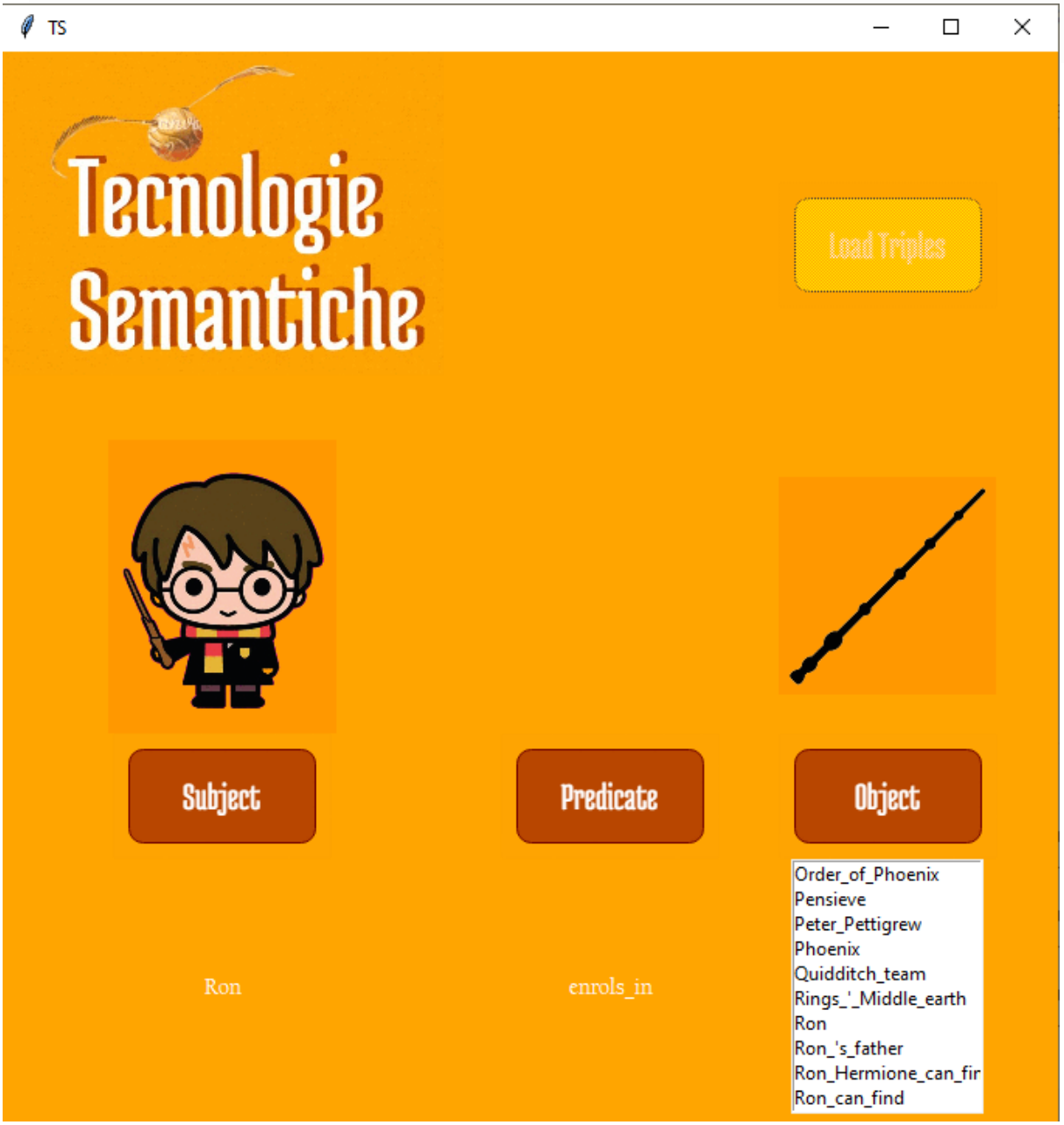
```
#Adding triple to neo4j graph
def add_to_graph():
    # create a neo4j backed Graph
    g = rdflib.Graph(store="Neo4j")

    # set the configuration to connect to your Neo4j DB
    theconfig = {'uri': "neo4j://127.0.0.1:7687",
                 'database': "neo4j",
                 'auth': {'user': "neo4j", 'pwd': "sas"}}

    g.open(theconfig, create = True)

    resource = rdflib.Namespace("http://dbpedia.org/")

    sub = rdflib.URIRef("http://dbpedia.org/resource#" + selected_sub)
    g.add((sub, rdflib.RDF.type, resource.subject))
    g.add((sub, resource.name, rdflib.Literal(selected_sub)))
    g.add((sub, resource.link, rdflib.Literal("http://dbpedia.org/resource/" + selected_sub)))

    pred = rdflib.URIRef("http://dbpedia.org/" + selected_pred)

    obj = rdflib.URIRef("http://dbpedia.org/resource#" + selected_obj)
    g.add((obj, rdflib.RDF.type, resource.object))
    g.add((obj, resource.name, rdflib.Literal(selected_obj)))
    g.add((obj, resource.link, rdflib.Literal("http://dbpedia.org/resource/" + selected_obj)))

    g.add((sub, pred, obj))
    print("Done!")
```

Inoltre è stata implementata un’interfaccia grafica che permettesse un uso più intuitivo del sistema. L’interfaccia è stata realizzata in linguaggio Python ed è stata integrata sia con Ampligraph che con neoSemantics. In particolare, si sono utilizzate le funzioni di ampligraph per caricare il modello addestrato e per il calcolo delle metriche della tripla specificata.

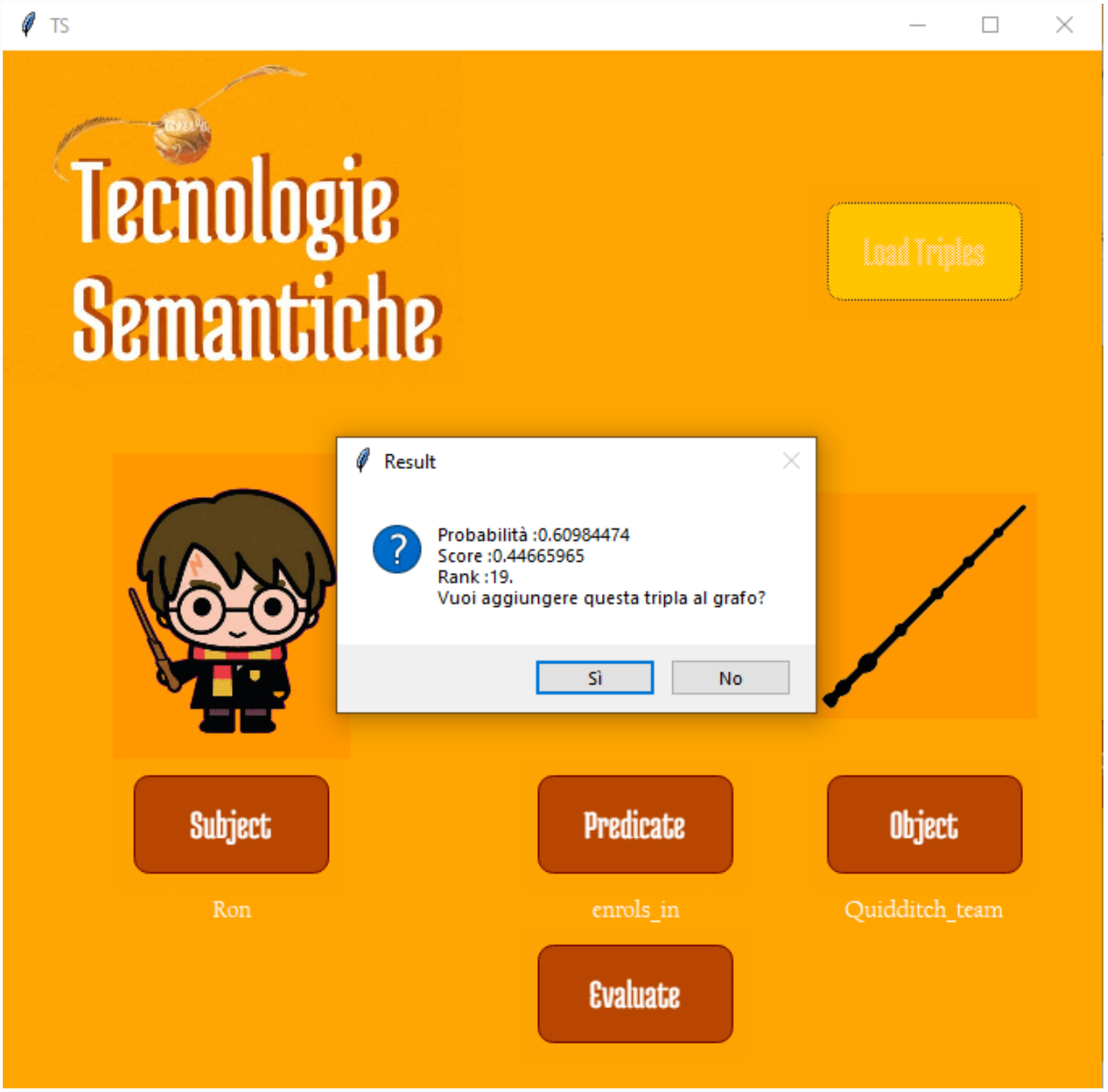
Funzionalità

In questo paragrafo sono riportate delle immagini illustrative delle funzionalità del sistema.

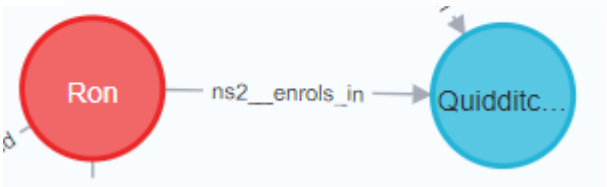
L’interfaccia permette di specificare una tripla selezionando soggetto, predicato e oggetto da una lista di possibili valori. Nell’esempio si sta selezionando la tripla “Ron enrols_in Quidditch_Team”, la quale ci si aspetta risulti veritiera in quanto Ron entra a far parte della squadra di Quidditch al quinto anno.

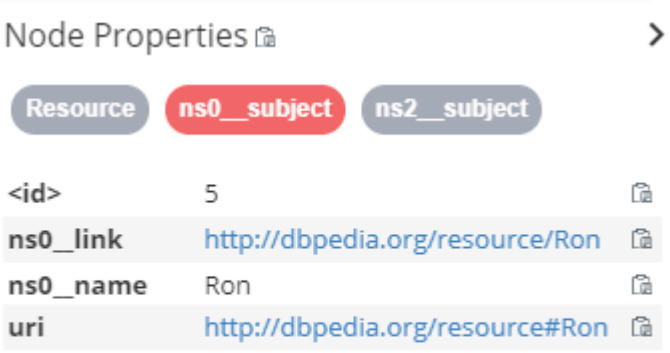


Una volta cliccato su “Evaluate” il sistema caricherà il modello addestrato ed effettuerà un’evaluation sulla tripla specificata. Al termine di quest’operazione, i risultati saranno mostrati a video e si chiede all’utente se vuole inserire la tripla nel grafo a cui è connesso. Nel nostro specifico caso di “Ron enrols_in Quidditch_Team” la probabilità è più o meno alta, per cui scegliamo di inserirla nel grafo.

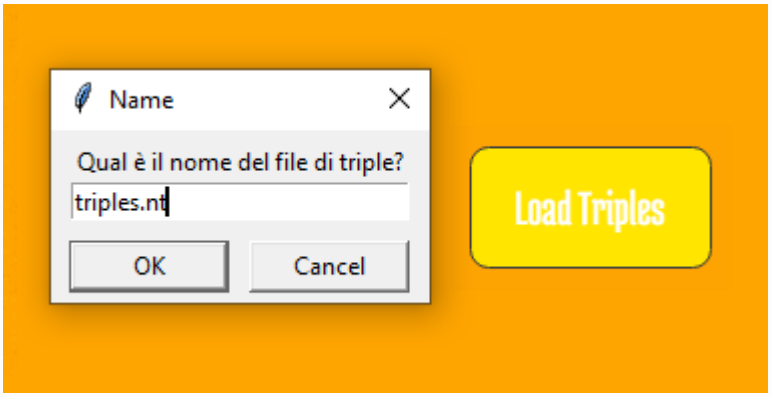


L’immagine qui riportata vuole verificare che la tripla sia stata effettivamente aggiunta al grafo. In particolare Ron è identificato come subject con proprietà:
- name : Ron
-link : http://dbpedia.org/resource/Ron





Infine si vuole testare se è possibile caricare nel grafo un intero file di triple. Cliccando sul pulsante “Load Triples” compare la finestra che chiede il nome del file .nt dalla quale si vogliono caricare le triple. Questo pulsante viene disattivato nel momento in cui si stanno selezionando triple singole per evitare eventuali conflitti, si riattiva in seguito alla valutazione della tripla selezionata.



L’immagine qui riportata vuole verificare che le triple siano state effettivamente aggiunte al grafo. In particolare, i soggetti sono evidenziati in rosso mentre gli oggetti in blu. In alcuni casi i soggetti fungono anche da oggetti, per cui saranno presenti più label associate.