
Università degli studi di Salerno
Big Data & Tecnologie Semantiche



Progettazione ontologia per eventi sismici con approccio machine learning



Membri del gruppo

Serino Valerio
Rolando Maxime
Maresca Liccione Antonio
Torre Flavio

Sommario

| | |
|--|------------------|
| <i>Introduzione</i> | <i>4</i> |
| <i>Modello concettuale</i> | <i>5</i> |
| <i>Data analysis.....</i> | <i>7</i> |
| <i>Machine learning.....</i> | <i>8</i> |
| MLP | 8 |
| Deep learning | 12 |
| Conclusioni machine learning..... | 14 |
| <i>Ontologia.....</i> | <i>15</i> |
| Scelte implementative..... | 16 |
| <i>Procedura applicativo.....</i> | <i>19</i> |
| <i>Strumenti di supporto</i> | <i>23</i> |
| <i>Conclusioni e sviluppi futuri.....</i> | <i>24</i> |

Abstract

In questo documento, viene presentata una nuova strategia per la raccolta di informazioni utili su eventi sismici predetti tramite approccio machine learning e tecnologie semantiche. L'assenza di soluzioni ontologiche che affrontano tale problematica, ha comportato l'esigenza di proporre una. Viene dunque illustrata l'architettura dell'applicativo ad alto livello, seguita dalla descrizione del dataset e delle diverse reti neurali implementate per la classificazione multiclasse degli eventi. Successivamente viene posta l'attenzione sull'implementazione dell'ontologia con i diversi metodi adottati, al fine di riusare progetti Linked Open Data già esistenti come dbpedia. Viene poi descritto in dettaglio il comportamento dell'applicativo e le varie tecnologie e librerie utilizzate. Infine, vengono esposte le conclusioni e i possibili sviluppi futuri dell'applicativo.

Introduzione

Allo stato attuale, le ontologie sono state riconosciute come una componente importante per la costruzione del Web semantico. Le ontologie sono concettualizzazioni di concetti e regole tramite il brainstorming con gli esperti collegati a un dominio specifico in accordo con gli scenari.

Sebbene oggi si stiano sviluppando ontologie di catastrofi in vari settori, la letteratura mostra che esiste ancora un divario considerevole nel dominio dei fenomeni sismici.

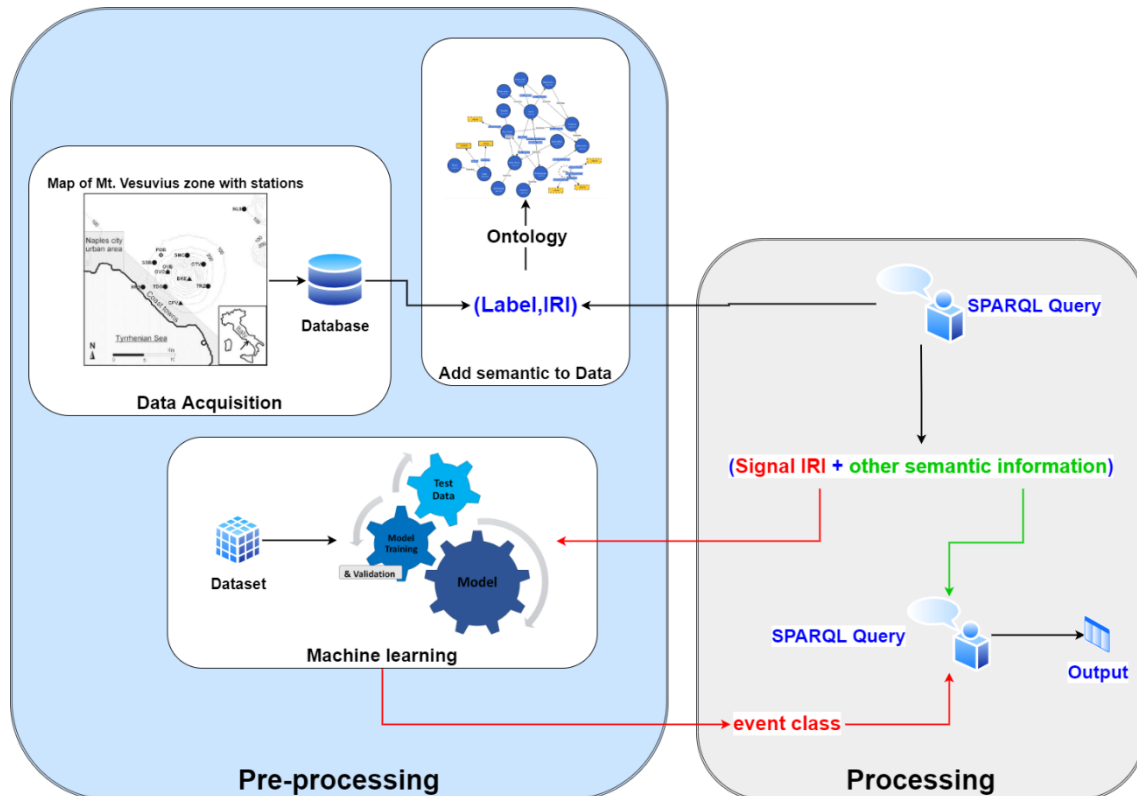
Il task che ci siamo posti è quello di dare una semantica a segnali sismici generici dopo aver predetto a quale particolare evento appartengano. L'implementazione del task è stata preceduta da uno studio di fattibilità che ha riguardato l'analisi di articoli scientifici e di modelli proposti da ricercatori per la previsione di fenomeni sismici (1) (2).

Data la mancanza di un'ontologia sulle catastrofi naturali, in particolare sui terremoti, è nata la necessità di definirne una. Come caso di studio sono stati analizzati segnali registrati da quattro stazioni selezionate dalla rete di monitoraggio del Mt. Vesuvio.

Lo sviluppo dell'applicativo avviene in due fasi: una prima in cui viene realizzata la classificazione multiclasse dei segnali, una seconda fase in cui viene aggiunta semantica ai segnali predetti. La classificazione distingue segnali sismici locali (causati dall'uomo o da sorgenti esterne), quali esplosioni in cava e tuoni, da terremoti vulcano-tettonici (fenomeni naturali).

Modello concettuale

Nella figura seguente viene mostrata la pipeline dell'applicativo:



Il modello proposto richiede una fase di *Pre-processing* in cui vengono eseguite le seguenti operazioni:

- per ogni segnale acquisito sul campo (*Data Acquisition*), viene istanziato un individuo all'interno dell'ontologia (*Add semantic to Data*), con i seguenti attributi:
 - *Label*, un identificativo scelto dall'utente
 - *IRI*, il path del segnale
- addestramento della rete neurale

La fase di *Processing* è invece, il core dell'applicativo e prevede i seguenti step:

- tramite *Label*, viene eseguita una SPARQL Query sull'ontologia per il recupero dell'*IRI* del segnale

- l'IRI del segnale viene utilizzato per ottenere il file contenente il segnale
- il file viene dato in input alla rete neurale
- l'output della rete neurale viene utilizzato per istanziare un individuo che rappresenta il particolare tipo di evento
- viene eseguita una SPARQL Query sull'ontologia per il recupero dell'evento predetto con l'aggiunta di informazioni

Data analysis

Il data set in possesso consiste di un totale di 908 segnali, appartenenti ad una delle seguenti classi/eventi: terremoti, scoppi in cava, tuoni:

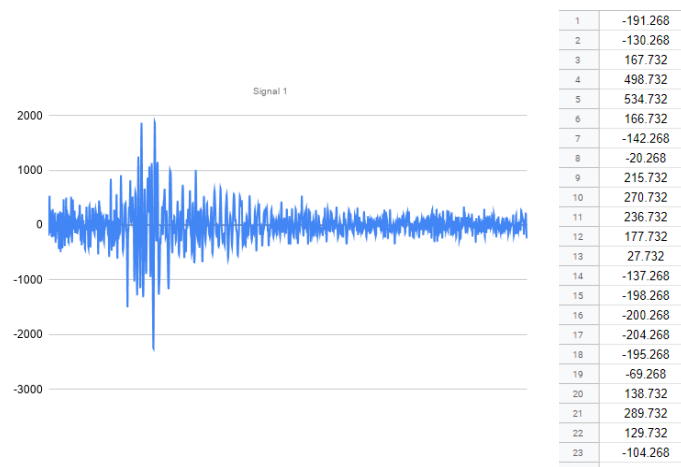
n. segnali registrati:

- 416 terremoti
- 464 scoppi in cava
- 28 tuoni

A causa di un numero inferiore di campioni del fenomeno ‘Tuoni’, il dataset risulta non bilanciato e la classificazione dunque meno accurata per tale classe.

Per ogni evento registrato, gli analisti hanno salvato un segnale lungo 20 secondi. Poiché la frequenza di campionamento è 100 Hz, ogni evento è composto da 2000 campioni, più 400 dovuti dal fenomeno dell’overlapping.

Nella seguente figura, viene mostrato un generico segnale nella sua versione analogica e digitale.



Machine learning

Per la discriminazione delle tre classi di eventi registrati in ogni stazione, sono stati utilizzati e messi a confronto due approcci di rete neurale:

- 1) *Multilayer perceptron (MLP)*
- 2) *Deep learning*

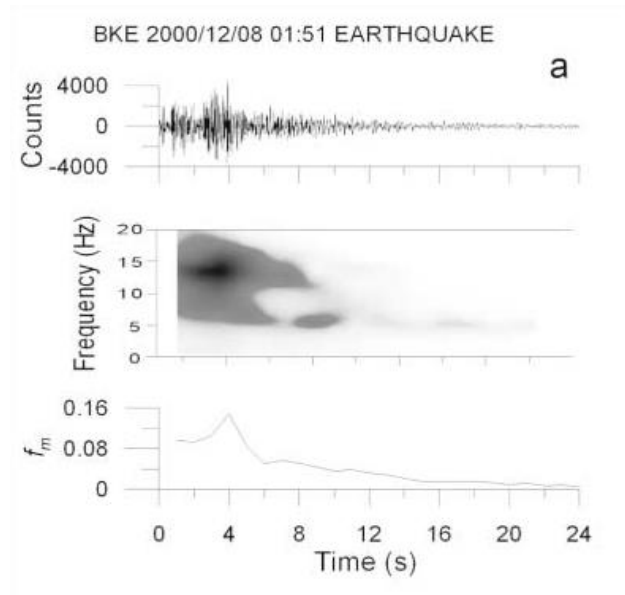
L'output della rete neurale per entrambi gli approcci restituisce il vettore contenente le probabilità associate alle rispettive classi.

Probabilità di corretta decisione:

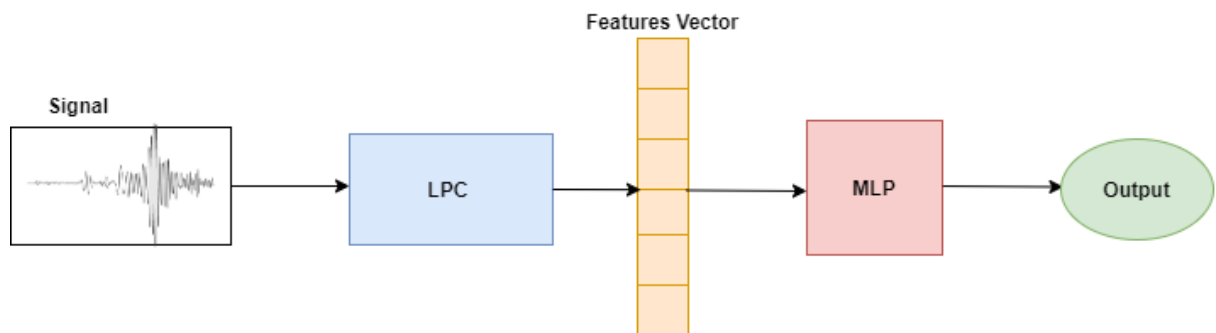
- *Terremoto* $\rightarrow [1,0,0]$
- *Scoppi in cava* $\rightarrow [0,1,0]$
- *Tuoni* $\rightarrow [0,0,1]$

MLP

L'implementazione dell'algoritmo MLP backpropagation è preceduta da una fase di pre-processing del dataset per l'estrazione delle features di interesse mediante l'algoritmo LPC (Linear Predictive coding). LPC è molto utilizzato per l'elaborazione dei segnali per rappresentare l'involuppo spettrale di un segnale numerico in forma compressa, utilizzando le informazioni provenienti da un modello predittivo lineare. [riferimento al paper] In figura è mostrato il risultato di un segnale prima e dopo l'estrazione delle features.



L'architettura MLP viene mostrata nella figura seguente:



La rete neurale Multi Layer Perceptron (MLP) proposta, utilizza l'algoritmo di addestramento backpropagation per l'aggiornamento dei pesi, calcolati con la regola del gradiente discendente.

Per l'addestramento della rete neurale, il 60% del dataset è stato usato per il training set, il 20% per il test set, il restante 20% per il validation test.

Definizione parametri di addestramento:

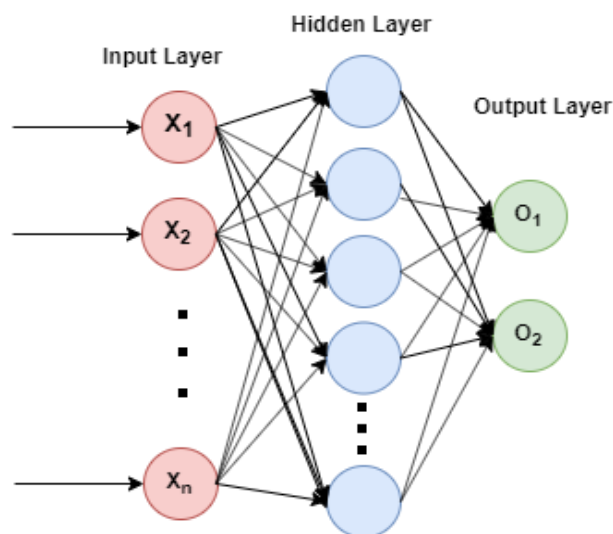
- ❖ hidden_layer_sizes= 42 (numero di neuroni e di hidden layer)
- ❖ batch_size=50 (numero di campioni per iterazione)

- ❖ `learning_rate_init=0.001` (tasso di addestramento)
- ❖ `momentum=0.9` (influenza direzione del gradiente)
- ❖ `max_iter=640` (numero massimo di iterazioni)
- ❖ `early_stopping=False` (interruzione precoce della rete per evitare l'overfitting)

La rete si suddivide in tre layer:

1. il layer di input con il features vector, ovvero le componenti spettrali del segnale contenenti le informazioni più rilevanti per la classificazione
2. un unico hidden layer tra input e output con 42 neuroni
3. un output layer con 2 neuroni data la classificazione per tre classi

Di seguito viene riportato la struttura interna della rete MLP:

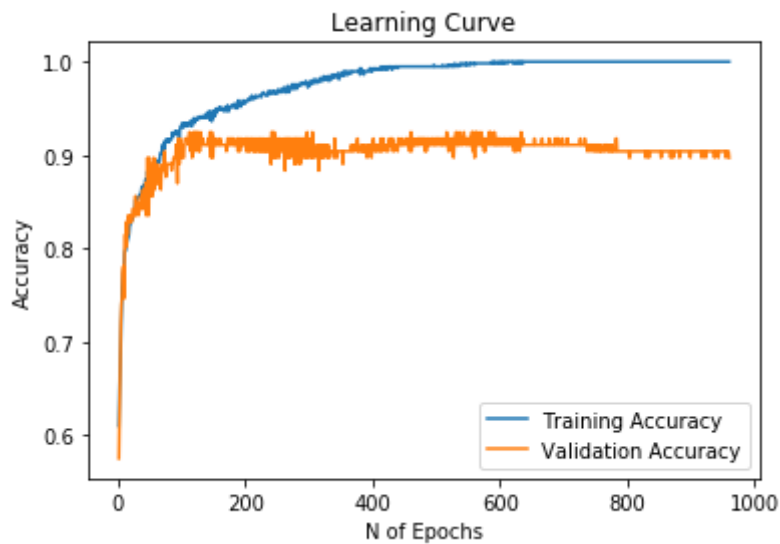


n (numero campioni dopo la features extraction) = 80

Performance della rete neurale:

- Training Accuracy 0.9793388429752066
- Test Accuracy 0.9560439560439561
- Validation Accuracy 0.8972602739726028

Di seguito vengono riportati i grafici di training e validation accuracy in funzione del numero di epoche:



Di seguito viene mostrata la matrice contenente le prediction sull'intero dataset per le seguenti etichette/classi: 0 terremoti, 1 scoppi in cava, 2 tuoni

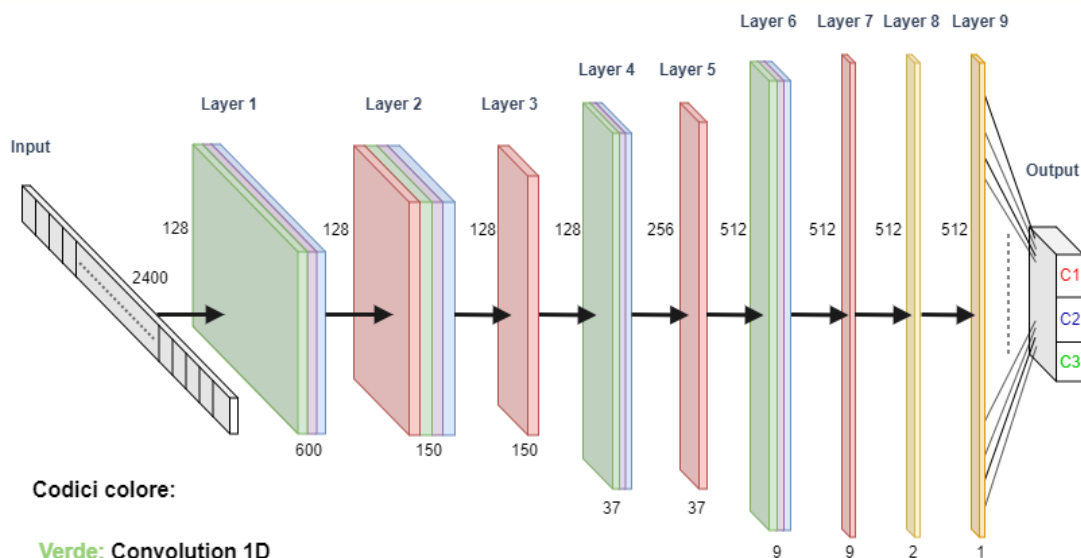
[illegible]

Deep learning

Per l'addestramento della rete neurale, 80% del dataset è stato usato per il training set, il restante 20% per il test set. (Dato che la dimensione del dataset contiene un numero esiguo di campioni per ogni classe di appartenenza, calcoliamo la validation usando solo il test set.)

Descrizione dei parametri di addestramento:

- ❖ `batch_size=128` (numero di campioni per iterazione)
- ❖ `epochs=200` (numero di epoche)
- ❖ `shuffle=True`
- ❖ `callbacks=[reduce_lr]`
 - `monitor='val_acc'` (quantità da monitorare)
 - `factor=0.5` (fattore di riduzione)
 - `patience=10` (numero massimo di epoche per la quale la quantità monitorata non cambia per la riduzione del learning rate)
 - `min_lr=0.0001` (valore minimale del learning rate)



Codici colore:

- Verde: Convolution 1D
- Viola: Batch Normalization
- Blu: Activation ("ReLU")
- Rosso: Max Pooling 1D
- Giallo: Lambda ("mean")
- Arancione: Dense ("SoftMax")

Descrizione livelli:

- Livello 1:
 - Conv1D (num_parametri: **10,368**)
 - BatchNorm (num_parametri: **512**)
 - Activation ('ReLU')
- Livello 2:
 - MaxPooling1D
 - Conv1D (num_parametri: **49,280**)
 - BatchNorm (num_parametri: **512**)
 - Activation ('ReLU')
- Livello 3:
 - MaxPooling1D
- Livello 4:
 - Conv1D (num_parametri: **98,560**)
 - BatchNorm (num_parametri: **1,024**)
 - Activation ('ReLU')
- Livello 5:
 - MaxPooling1D
- Livello 6:
 - Conv1D (num_parametri: **393,728**)
 - BatchNorm (num_parametri: **2,048**)
 - Activation ('ReLU')
- Livello 7:
 - MaxPooling1D
- Livello 8:
 - Lambda ('Mean')
- Livello 9:
 - Dense('SoftMax') (num_parametri: **1,528**)

Total parametri: **557,571**

Parametri addestrabili: **555,523**

Parametri non addestrabili: **2,048**

Performance della rete neurale:

Training Accuracy: 1.0

Testing Accuracy: 0.9505494485844622

Conclusioni machine learning

La rete MLP usata ha prodotto test accuracy minore del 90%; in seguito, come illustrato precedentemente, aggiungendo a tale rete una fase preliminare di feature extraction, le performance sono risultate simili all'approccio deep-learning.

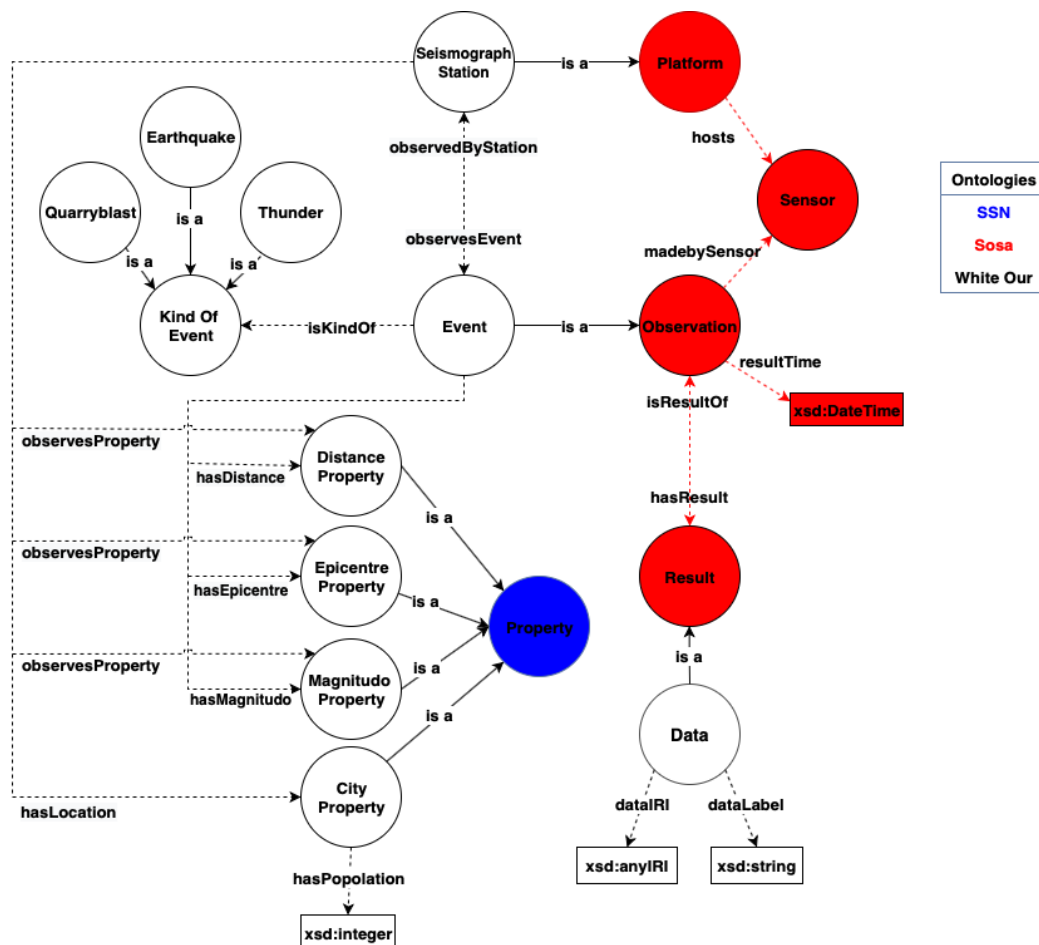
In conclusione, si è preferito l'utilizzo della rete deep per i seguenti motivi:

- gli input sono segnali grezzi, dunque non occorre una fase di preelaborazione delle feature
- migliori performance per dataset di grandi dimensioni

Ontologia

L'ontologia implementata, definita come Event Evaluation Volcano (EEV), permette il recupero di informazioni desiderate, al fine di avere una migliore comprensione dei fenomeni sismici in esame.

Sono stati riusati alcuni concetti presenti in ontologie come SSN e SOSA e altri definiti ad hoc per il raggiungimento del nostro task. L'ontologia Semantic Sensor Network (SSN) è stata scelta poiché presenta entità e relazioni per descrivere i sensori e le loro osservazioni, le procedure coinvolte, i campioni utilizzati, le proprietà osservate e gli attuatori. SSN include inoltre, un'ontologia di base leggera ma autonoma chiamata SOSA (Sensore, Osservazione, Campione e Attuatore) per le sue classi e proprietà elementari. Infatti, nell'analisi del dominio (fenomeni sismici), sono state identificate 11 feature di interesse (entità) e proprietà correlate:



La figura sopra riportata mostra come la *EEV ontology* si colleghi a *SSN* e *SOSA*:

le entità *Event*, *Kind Of Event* e classi figlie descrivono la classificazione dei fenomeni sismici e sono connessi all'entità preesistente *Observation* di *SOSA*;

Distance, *Epicentre*, *Magnitudo Property* rappresentano le caratteristiche di un generico evento sismico (possibili sviluppi futuri, quali i collegamenti ad ontologie per la definizione di misure e/o coordinate geografiche come *GeoNames* e *Quantities Kinds and Units ontology (QU)*, ci hanno spinto a definirle come entità e non semplici dataProperty/attributi della classe *Event*);

Sismograph Station identifica quale stazione ha registrato un particolare evento tramite i propri sensori; la classe è dunque collegata all'entità *Event* tramite le proprietà *observedByStation* e *observesEvent* e, specializza la classe *Platform (SOSA)* al fine di riutilizzare la classe *Sensor (SOSA)*;

City Property localizza la città in cui è situata la generica stazione;

infine, l'entità *Data* individua il segnale registrato; sono state definite le dataProperty *dataIRI* e *dataLabel*.

Scelte implementative

La *EEV* è stata implementata su Protégé, un editor ontologico gratuito e open source, che fornisce un'interfaccia utente grafica per definire le ontologie. Sono state importate le ontologie *SSN* e *SOSA* come analizzato in precedenza;

Al fine di avere un'ontologia più consistente e di recuperare più informazioni possibili da ontologie note, si è posto il problema di recuperare individui per l'entità *City Property* e di collegare questi a *dbpedia*.

Un primo tentativo è stato quello di importare l'ontologia *dbpedia* su Protégé; a causa delle grosse dimensioni dei suoi dataset, si è preferito adottare le seguenti soluzioni.

Per il popolamento della classe *City Property*, gli individui sono stati recuperati dalla seguente query sul dataset di *dbpedia*, tramite *Virtuoso* (editor per le SPARQL query):

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
```

```
SELECT DISTINCT ?name ?population
WHERE {
    ?city a dbo:City;
    dbo:country dbr:Italy.
    ?city rdfs:label ?name. # non voglio IRI
    FILTER (langMatches(lang(?name), "it")).

    OPTIONAL{?city dbo:populationTotal ?population}.
    FILTER (xsd:integer(?population))
}
order by asc(str(?name))
```

La SPARQL query restituisce l'elenco di tutte le province d'Italia, con le relative popolazioni, in ordine alfabetico. L'output viene salvato in un file csv e caricato nella nostra ontologia EEV tramite plug-in Callfie di Protégé tramite le seguenti regole:

```
{
  "Collections": [
    {
      "sheetName": "dati",
      "startColumn": "A",
      "endColumn": "B",
      "startRow": "1",
      "endRow": "+",
      "comment": "",
      "rule": "Individual: @A*(xsd:string)\nTypes: City_Property\nFacts: hasPopulation @B*(xsd:integer)",
      "active": true
    }
  ]
}
```

Lo script permette di creare un'istanza di *City_Property* per ogni riga presente nella colonna A, associandole l'attributo *hasPopulation* con relativo valore della colonna B. Per collegare le istanze create a quelle presenti in dbpedia, è stato cambiato il namespace in <http://dbpedia.org/resource#CityName>.

Procedura applicativo

Nel seguito, viene illustrato la procedura dell'applicativo sviluppato:

1) caricamento del modello e dei pesi della rete neurale

```
from keras.models import load_model
os.chdir('/content/drive/My Drive/Machine Learning/BigDataProject/modelML')
model=load_model('seismologicML.model')
model.load_weights('seismologicML.weights')
model.summary()
```

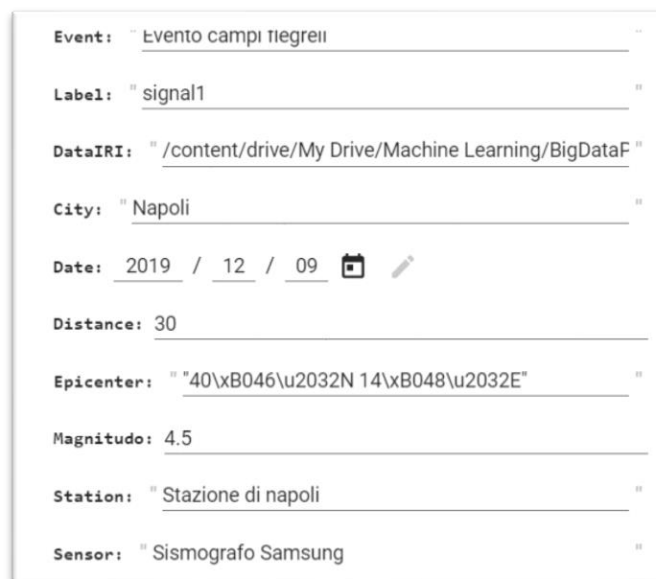
2) caricamento dell'ontologia

```
path = 'file:///content/drive/My Drive/Machine Learning/BigDataProject/eevOntology.owl'
onto = get_ontology(path).load()
```

3) creazione del form per i dati in input

```
{ display-mode: "form" }

Event = "Evento campi flegreii" #@param {type:"string"}
Label = "signal1" #@param {type:"string"}
DataIRI = "/content/drive/My Drive/Machine Learning/BigDataProject/signal1.dat" #@param {type:"string"}
City = "Napoli" #@param {type:"string"}
Date = "2019-12-09" #@param {type:"date"}
Distance = 30#@param {type:"number"}
Epicenter = "40\xB046\u2032N 14\xB048\u2032E" #@param {type:"string"}
Magnitudo = 4.5#@param {type:"number"}
Station = "Stazione di napoli" #@param {type:"string"}
Sensor = "Sismografo Samsung" #@param {type:"string"}
```



The screenshot shows a web form with the following fields and values:

- Event: Evento campi flegreii
- Label: signal1
- DataIRI: /content/drive/My Drive/Machine Learning/BigDataProject/signal1.dat
- City: Napoli
- Date: 2019 / 12 / 09 (with a calendar icon)
- Distance: 30
- Epicenter: 40\xB046\u2032N 14\xB048\u2032E
- Magnitudo: 4.5
- Station: Stazione di napoli
- Sensor: Sismografo Samsung

4) Esempio di creazione delle istanze in base ai dati inseriti dall'utente nel form.

```
# creazione istanza 'event' nell'ontologia con label dato dell'utente
event1 = onto.Event(Event)

# creazione istanza 'data' che contiene il segnale
data1 = onto.Data(Label)

# aggiunta attributi all'oggetto data1 --> (label,IRI)
data1.dataIRI=[DataIRI]

#collegamento event1 con data1
event1.hasResult = [data1]
data1.isResultOf = [event1]

#collegamento event1 con magnitudo
magnitudoObject = onto.Magnitudo_Property(Label + "EventMagnitudo")
magnitudoObject.hasMagnitudoValue = [Magnitudo]
event1.has_magnitudo = [magnitudoObject]
```

5) Query per il recupero delle informazioni dell'evento dato il label

```
for row in g.query("""
PREFIX : <http://www.semanticweb.org/BigDataProject/ontologies/2019/10/eev#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?o ?event ?station ?sensor ?city ?distanceValue ?epicenterValue ?magnitudoValue ?date
WHERE {
  ?s :dataLabel ""+"\""+Label+"\""+\"^^xsd:string .
  ?s :dataIRI ?o .
  ?s :sosa:isResultOf ?event .
  ?station :observe_event ?event .
  ?station :sosa:hosts ?sensor .
  ?station :has_location ?city .
  ?event :has_distance ?distance .
  ?event :has_epicenter ?epicenter .
  ?event :has_magnitudo ?magnitudo .
  ?event :sosa:resultTime ?date .
  ?distance :hasDistanceValue ?distanceValue .
  ?epicenter :hasEpicenterValue ?epicenterValue .
  ?magnitudo :hasMagnitudoValue ?magnitudoValue .
}"""):
IriX=row.o
EventX=re.sub(r'.*#', "", row.event)
StationX=re.sub(r'.*#', "", row.station)
SensorX=re.sub(r'.*/', "", row.sensor)
CityX=re.sub(r'.*/', "", row.city)
DistanceX=re.sub(r'.*#', "", row.distanceValue)
EpicenterX=re.sub(r'.*#', "", row.epicenterValue)
MagnitudoX=re.sub(r'.*#', "", row.magnitudoValue)
DateTimeX=re.sub(r'.*#', "", row.date)
```

6) Query che fornisce l'uri della risorsa di Dbpedia associata alla proprietà city.

```
sparql = SPARQLWrapper("http://dbpedia.org/sparql") # wrapper per virtuoso
sparql.setQuery("""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

    SELECT DISTINCT ?city
    FROM <http://dbpedia.org>
    WHERE {
        ?city rdfs:label """"+"\""+CityX+""""@"it .
    }
""")
sparql.setReturnFormat(JSON)
results = sparql.query().convert()

for result in results["results"]["bindings"]:
    print(result["city"]["value"])

# salvataggio risultati query
city= result["city"]["value"]
```

7) Query restituisce da Dbpedia la popolazione e le immagini associate alla città

```
sparql.setQuery("""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX dbr: <http://dbpedia.org/resource/>
    PREFIX dbp: <http://dbpedia.org/property/>
    PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

    SELECT DISTINCT ?quartiere
    WHERE {
        ?x dbp:seat <"""+city+"> '+'""".
        ?quartiere dbo:locatedInArea|dbo:province ?x.
    }
    LIMIT 6
""")
# la query ritorna tutti i comuni e zone rurali che sono presenti nell'area metropoli-
tana di Napoli
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
quartieri=[]
for result in results["results"]["bindings"]:
    quartiere = re.sub(r'.*/','"',result["quartiere"]["value"]) # stampa migliore
    quartieri.append(quartiere)
    print(quartiere)
```

8) Definizione della funzione per la stampa di informazioni sul terremoto

```
def outputEarthquake():
    print('/!\ \ /!\ \ ALLERTA TERREMOTO /!\ \ /!\ \ \n')
    print('Avvertimento per '+EventX+'di cittadini nella provincia di '+CityX+
          '\n_____ \n'+
          ' \nZone a rischio:\n')

    for x in quartieri:
        print('-----')
        print(x)
    print('_____')
    print('\nCaratteristiche evento \n')
    print('Registrato da: '+StationX+'\n'+
          'Sensore: '+SensorX+'\n'+
          'In data: '+DateTimeX+'\n'+
          'Epicentro: '+EpicenterX+'\n'+
          'Magnitudo: '+MagnitudoX+'\n'+
          'Distanza: '+DistanceX)
```

9) Fase di predizione del tipo di evento e output nel caso di rilevazione di terremoto.

```
import numpy as np

signal1 = np.loadtxt(IriX)
signal1 = np.asarray(signal1)

signal1 = np.reshape(signal1, (1,signal1.shape[0],1))

pre=model.predict(signal1).astype(int)[0]# pre.astype(int)[0]-->predno primo vettore

if(np.array_equal(np.asarray([1,0,0]),pre)):
    # creazione istanza 'Earthquake' nell'ontologia e assegnazione all'evento1
    event1.is_kind_of = [earthquakeObject]
    outputEarthquake()
elif(np.array_equal(np.asarray([0,1,0]),pre)):
    # creazione istanza 'QuarryBlast' nell'ontologia e assegnazione all'evento1
    event1.is_kind_of= [quarryBlastObject]
    print(event1.is_kind_of)
    print(onto.QuarryBlast1)
elif(np.array_equal(np.asarray([0,0,1]),pre)):
    # creazione istanza 'Thunder' nell'ontologia e assegnazione all'evento1
    event1.is_kind_of = [thunderObject]
    print(event1.is_kind_of)
    print(onto.Thunder1)
else:
    print("NO CLASSIFYING")
```

/!\ /!\ ALLERTA TERREMOTO /!\ /!\

Avvertimento per Evento campi flegreiidi cittadini nella provincia di Napoli

Zone a rischio:

Mount_Somma

Mount_Barbaro

Mount_Vesuvius

Ischia

Mariglianella

Ischia,_Campania

Caratteristiche evento

Registrato da: Stazione di napoli

Sensore: Sismografo Samsung

In data: 2019-12-19

Epicentro: 40°46'N 14°48'E

Magnitudo: 4.5

Distanza: 30

Strumenti di supporto

Per la realizzazione dell'ontologia sono stati usati i seguenti tool:

- Protegè, con plug-in Callfie per la definizione dell'ontologia
- Virtuoso, per SPARQL query a dbpedia

Librerie di supporto:

- Keras, TensorFlow per la realizzazione delle reti neurali
- Audiolazy per l'estrazione delle feature per MLP
- Owlready2 per importare e gestire l'ontologia
- Rdfliib per interrogare l'ontologia
- SPARQLWrapper per interrogare dbpedia

Gli script sono stati implementati usando Python3 sulla piattaforma Google Colab.

Conclusioni e sviluppi futuri

L'implementazione dell'applicativo ha richiesto la conoscenza di RDF, OWL, SPARQL, degli approcci machine learning, lo studio di ontologie esterne come SSN, SOSA e DBPEDIA, librerie come owlready2, SPARQLWrapper, rdflib e il tool Protégè.

In conclusione, tutti gli obiettivi preposti all'inizio del progetto sono stati raggiunti. È stata creata un'ontologia per arricchire di informazioni i fenomeni sismici considerati ed è stato creato un classificatore per determinarne la loro natura. Il nostro lavoro, inoltre, può rappresentare un punto di partenza per lo sviluppo di sistemi più complessi.

Per quanto riguarda possibili sviluppi futuri, si può prevedere di espandere l'ontologia realizzando dei collegamenti con GeoNames al fine di riusare concetti di misurazioni e coordinate geografiche.

Link repository github:

<https://github.com/vale-29/BigData.git>