

# Knowledge Graph

# Lecture Outline

- Tools for Knowledge Graph
  - NLP tool: SPACY
  - Relation extraction: OpenIE
  - Managing graph data: NEO4J

# Spacy: Overview

- Spacy is a NLP toolkit based on Deep Learning
- Industry standard
- Very fast processing
- Python APIs

# Statistical Models

- **en\_core\_web\_sm**: English multi-task CNN, trained on OntoNotes
- **en\_core\_web\_md**: English multi-task CNN trained on OntoNotes, with Glove vectors trained on Common Crawl
- **en\_core\_web\_lg**: English multi-task CNN trained on OntoNotes, with GloVe vectors trained on bigger Common Crawl

# Spacy: Importing a model

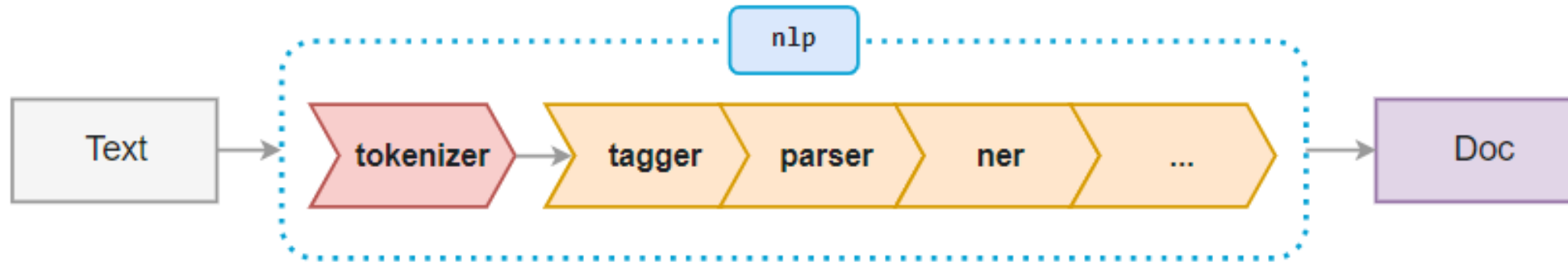
- General syntax:

*`spacy.load('model_name')`*

- Example:

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

# Processing Pipeline



Source: <https://spacy.io/usage/processing-pipelines>

# Creating an NLP object

```
import spacy
nlp = spacy.load('en_core_web_sm')

doc = nlp("He went to play basketball")
```

# Activating/disabling pipeline component

- Knowing the active components

*nlp.pipe\_names*

- Disabling components

*nlp.disable\_pipes('tagger', 'parser')*

# Part-of-Speech (POS) Tagging using spaCy

```
import spacy
nlp = spacy.load('en_core_web_sm')

# Create an nlp object
doc = nlp("He went to play basketball")

# print tokens and its part-of-speech tag
for token in doc:
    print(token.text, "-->", token.pos_)
```

## **Output:**

```
He -> PRON
went -> VERB
to -> PART
play -> VERB
basketball -> NOUN
```

## **To know the meaning of the tags:**

```
spacy.explain("PART")
```

# Dependency Parsing using spaCy

```
import spacy
nlp = spacy.load('en_core_web_sm')

# Create an nlp object
doc = nlp("He went to play basketball")

# print tokens and dependency
for token in doc:
    print(token.text, "-->", token.dep_)
```

## **Output:**

```
He -> nsubj
went -> ROOT
to -> aux
play -> advcl
basketball -> dobj
```

**To know the meaning of the tags:**

```
spacy.explain("nsubj")
```

# Named Entity Recognition using spaCy

```
import spacy
nlp = spacy.load('en_core_web_sm')

# Create an nlp object
doc = nlp("Indians spent over $71 billion on clothes in 2018")

# print tokens and ner tags
for ent in doc.ents:
    print(ent.text, ent.label_)
```

## **Output:**

Indians NORP  
over \$71 billion MONEY  
2018 DATE

## **To know the meaning of the tags:**

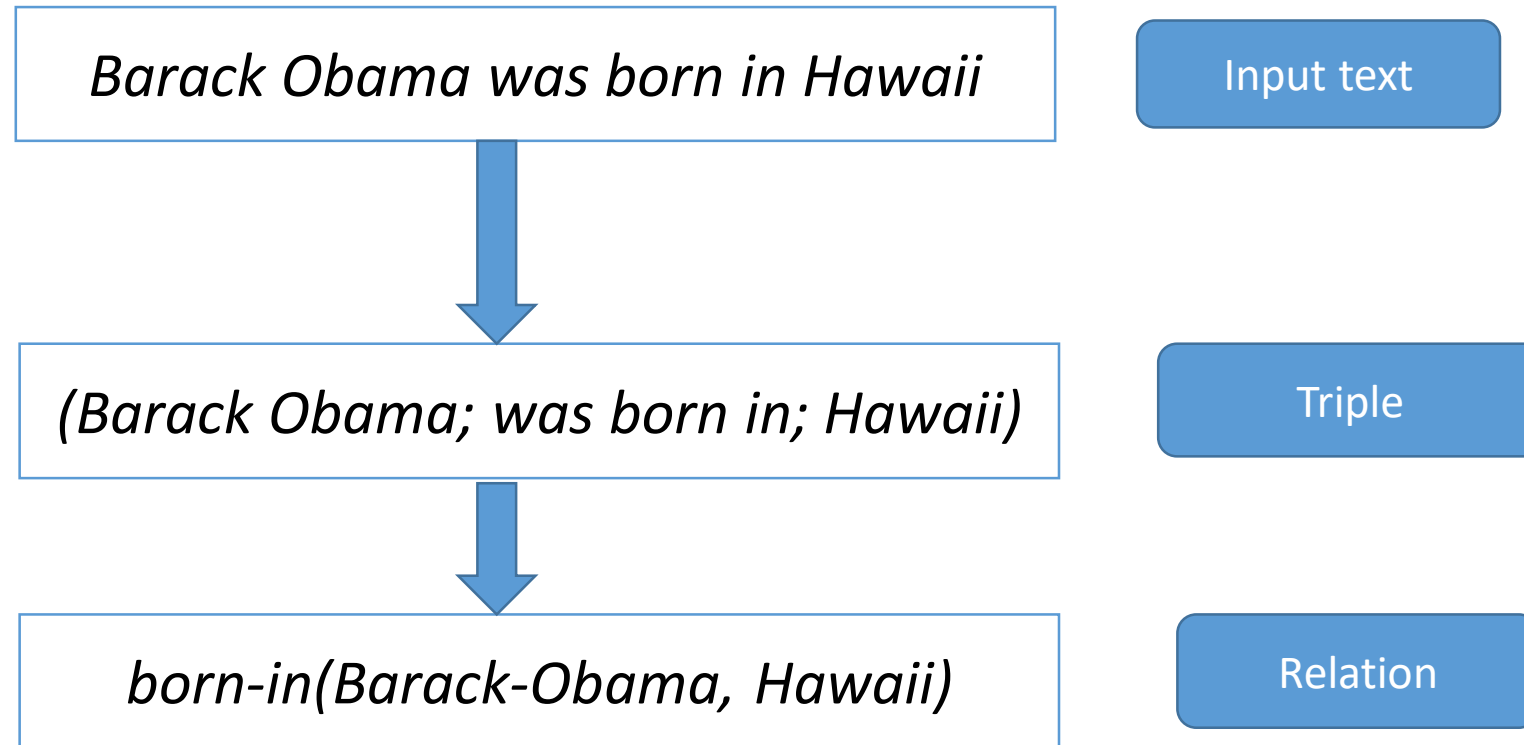
```
spacy.explain("NORP")
```

# Relation Extraction

# Open relation extraction: OpenIE

- Extraction of relation tuples, typically binary relations, from plain text

- Example:



# Running OpenIE

```
java -mx1g -cp "*" edu.stanford.nlp.naturalli.OpenIE /path/to/file1 /path/to/file2
```

*Files contain the senetences.*

Reference: <https://nlp.stanford.edu/software/openie.html>

# Managing Graph Data

NEO4J

# Main building blocks

- Nodes
- Relationships
- Properties

# Graph Data Model

- The model represents data in Nodes, Relationships and Properties
- Properties are key-value pairs
- Relationships have directions: Unidirectional and Bidirectional
- Each Relationship contains "Start Node" or "From Node" and "To Node" or "End Node"
- Both Nodes and Relationships contain properties
- Relationships connect nodes

# Cypher Query Language (CQL)

- Query language for Neo4j Graph Database.
- Declarative pattern-matching language.
- Follows SQL like syntax.
- Syntax is very simple and in human readable format.

# CQL Clauses for Reading

Sr.No	Read Clauses	Usage
1	MATCH	This clause is used to search the data with a specified pattern.
2	OPTIONAL MATCH	This is the same as match, the only difference being it can use nulls in case of missing parts of the pattern.
3	WHERE	This clause id is used to add contents to the CQL queries.
4	START	This clause is used to find the starting points through the legacy indexes.
5	LOAD CSV	This clause is used to import data from CSV files.

# CQL Clauses for Writing

Sr.No	Write Clause	Usage
1	CREATE	This clause is used to create nodes, relationships, and properties.
2	MERGE	This clause verifies whether the specified pattern exists in the graph. If not, it creates the pattern.
3	SET	This clause is used to update labels on nodes, properties on nodes and relationships.
4	DELETE	This clause is used to delete nodes and relationships or paths etc. from the graph.
5	REMOVE	This clause is used to remove properties and elements from nodes and relationships.
6	FOREACH	This class is used to update the data within a list.

# CQL: General Clauses

Sr.No	General Clauses	Usage
1	RETURN	This clause is used to define what to include in the query result set.
2	ORDER BY	This clause is used to arrange the output of a query in order. It is used along with the clauses <b>RETURN</b> or <b>WITH</b> .
3	LIMIT	This clause is used to limit the rows in the result to a specific value.
4	SKIP	This clause is used to define from which row to start including the rows in the output.
5	WITH	This clause is used to chain the query parts together.
6	UNWIND	This clause is used to expand a list into a sequence of rows.
7	UNION	This clause is used to combine the result of multiple queries.
8	CALL	This clause is used to invoke a procedure deployed in the database.

# CQL: Data Types

Sr.No	CQL Data Type	Usage
1	Boolean	It is used to represent Boolean literals: true, false.
2	byte	It is used to represent 8-bit integers.
3	short	It is used to represent 16-bit integers.
4	int	It is used to represent 32-bit integers.
5	long	It is used to represent 64-bit integers.
6	float	It is used to represent 32-bit floating-point numbers.
7	double	It is used to represent 64-bit floating-point numbers.
8	char	It is used to represent 16-bit characters.
9	String	It is used to represent Strings.

# CQL: Comparison Operators

Sr.No	Boolean Operators	Description
1	=	It is a Neo4j CQL "Equal To" operator.
2	< >	It is a Neo4j CQL "Not Equal To" operator.
3	<	It is a Neo4j CQL "Less Than" operator.
4	>	It is a Neo4j CQL "Greater Than" operator.
5	<=	It is a Neo4j CQL "Less Than Or Equal To" operator.
6	> =	It is a Neo4j CQL "Greater Than Or Equal To" operator.

# CQL: Boolean Operators

Sr.No	Boolean Operators	Description
1	AND	It is a Neo4j CQL keyword to support AND operation. It is like SQL AND operator.
2	OR	It is a Neo4j CQL keyword to support OR operation. It is like SQL AND operator.
3	NOT	It is a Neo4j CQL keyword to support NOT operation. It is like SQL AND operator.
4	XOR	It is a Neo4j CQL keyword to support XOR operation. It is like SQL AND operator.

# Creating Nodes and labels

- Create nodes
  - CREATE (node1),(node2)
- Create nodes with label
  - CREATE (Obama:politician)
- Create node with multiple label
  - CREATE (Obama:politician:person)

# Creating Node with Properties

- Properties are given in key, value pairs
- Key -> property name, value-> value of the property
- Example:
  - `CREATE (Obama:politician{name: "Barack Obama", nationality:"American"})`

# Creating Relationships

- Note: relationships are directional (node1 -> node2)
- Step 1: create two nodes/entities
- Step 2: create relationship
- Example
  - CREATE (Obama:politician{name: "Barack Obama", nationality:"american"})
  - CREATE (Usa:Country {name: "USA"})
  - CREATE (Obama)-[r:CITIZEN\_OF]->(Usa)

# For More Details

- See these resources
  - NEO4J homepage: <https://neo4j.com/>
  - Book: [https://go.neo4j.com/rs/710-RRC-335/images/Graph Databases for Beginners.pdf](https://go.neo4j.com/rs/710-RRC-335/images/Graph_Databases_for_Beginners.pdf)