In this project I would like to automatically map metadata of an data set with an existing domain specific ontology. Quite a lot research has been done in the area of the ontology mapping and machine learnings techniques. The topic is very broad but it is not hard to find a very practical application of these high level theories.

The area I would like to address is the new European legislation, General Data Protection Regulation(GDPR). According to GDPR any company treating data about European citizens needs to comply with certain regulations. It needs to keep track of the data, it needs to answer the question about how and in what purpose it is used, it needs on any request from data subject provide a copy of the data and provide the right to be forgotten. Any fallacy in following the regulations results in severe fines.

It poses a number of challenges for data professional and companies processing personal data. The definition of personal data is much wider than traditional PII definition and might vary between domains. Democratic access to the data, creating/acquiring new data sets should not be caught up some bureaucratic procedure.

I would therefore find a method for automated detection of personal data in new data sets and sources. The challenge posed by EU might be new but the problem itself is not. Mapping between ontologies and metadata has been addressed before and I will rely on previous research, although limiting the scope to GDPR compliance.

In my eyes GDPR and definition of personal data is a form on domain specific ontology and that needs to be automatically mapped to new data sets and sources. This very pragmatic approach, although somewhat simplistic, does not close the door to wider usage of ontology mappings.

What needs to be done?

I will concentrate on the automation part of the project. What interests me the most is to find methods of automated handling of ontologies and possibly come up with machine learning algorithms to map metadata with ontologies. I will not touch upon defining GDPR compliant personal data ontology but rather keep a more generic PoC approach.

I will start by looking at WordNet. Although WordNet is not an ontology it provides a vocabulary and structure that is necessary to build an ontology.

WordNet is a lexical database for the English language and can be thought of as a combination of a thesaurus and a dictionary, with relationships between groups of words. There are two main categories in WordNet, a synset, and a lemma.

Lemmas are are the canonical root of a word. E.g. swim, swims, swam, and swimming all have the same root ‘swim’, and thus would be represented in WordNet as ‘swim’.

Synsets are groups of lemmas that could be considered interchangeable. For example, we might look up the synset ‘wedding.n’ and our first synset contains a group of lemmas “wedding, wedding ceremony, nuptials, hymeneals”. There also exists a few synsets with wedding as a verb, the first of which contains the following lemmas: “marry, get married, wed, conjoin, hook up with, get hitched with, espouse”.

Some basic concepts from Vocabulary.com

synset: a set of one or more synonyms

holonym: a word that names the whole of which a given word is a part

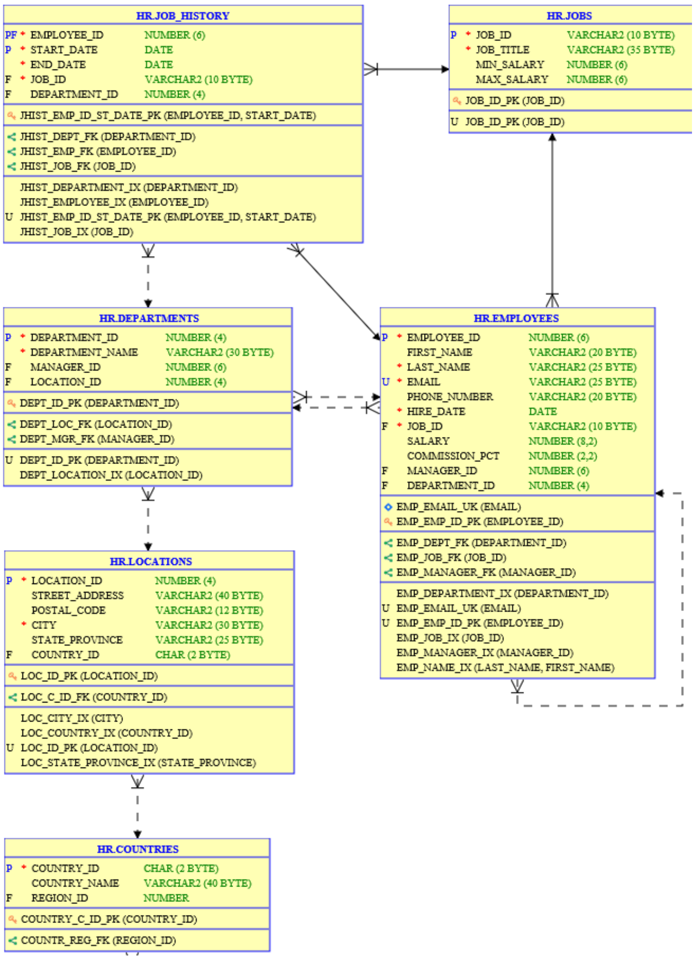
hyponym: a word that is more specific than a given word

hypernym: a word that is more generic than a given word

lemma: the heading that indicates the subject of an annotation or a literary composition or a dictionary entry

In addition to storing synsets and lemmas, WordNet also contains relationships between Synsets. For example, hypernyms and hyponyms constitute an "IS_A" relationship between two synsets (hypernyms being parents, and hyponyms being children). E.g. Employee is a hypernym of clerk, which in turn is a hyponym of Person.

After building an ontology containing synsets it should be possible to map a specific concept to synsets. In this simple example of a HR database it would be fairly easy to identify entities related to personal data.



HR.JOB_HISTORY

PF	*	EMPLOYEE_ID	NUMBER (6)
P	*	START_DATE	DATE
	*	END_DATE	DATE
F	*	JOB_ID	VARCHAR2 (10 BYTE)
F		DEPARTMENT_ID	NUMBER (4)
🔑		JHIST_EMP_ID_ST_DATE_PK (EMPLOYEE_ID, START_DATE)	
<		JHIST_DEPT_FK (DEPARTMENT_ID)	
<		JHIST_EMP_FK (EMPLOYEE_ID)	
<		JHIST_JOB_FK (JOB_ID)	
		JHIST_DEPARTMENT_IX (DEPARTMENT_ID)	
		JHIST_EMPLOYEE_IX (EMPLOYEE_ID)	
U		JHIST_EMP_ID_ST_DATE_PK (EMPLOYEE_ID, START_DATE)	
		JHIST_JOB_IX (JOB_ID)	

HR.JOBS

P	*	JOB_ID	VARCHAR2 (10 BYTE)
	*	JOB_TITLE	VARCHAR2 (35 BYTE)
		MIN_SALARY	NUMBER (6)
		MAX_SALARY	NUMBER (6)
🔑		JOB_ID_PK (JOB_ID)	
U		JOB_ID_PK (JOB_ID)	

HR.DEPARTMENTS

P	*	DEPARTMENT_ID	NUMBER (4)
	*	DEPARTMENT_NAME	VARCHAR2 (30 BYTE)
F		MANAGER_ID	NUMBER (6)
F		LOCATION_ID	NUMBER (4)
🔑		DEPT_ID_PK (DEPARTMENT_ID)	
<		DEPT_LOC_FK (LOCATION_ID)	
<		DEPT_MGR_FK (MANAGER_ID)	
U		DEPT_ID_PK (DEPARTMENT_ID)	
		DEPT_LOCATION_IX (LOCATION_ID)	

HR.EMPLOYEES

P	*	EMPLOYEE_ID	NUMBER (6)
		FIRST_NAME	VARCHAR2 (20 BYTE)
	*	LAST_NAME	VARCHAR2 (25 BYTE)
U	*	EMAIL	VARCHAR2 (25 BYTE)
		PHONE_NUMBER	VARCHAR2 (20 BYTE)
	*	HIRE_DATE	DATE
F	*	JOB_ID	VARCHAR2 (10 BYTE)
		SALARY	NUMBER (8,2)
		COMMISSION_PCT	NUMBER (2,2)
F		MANAGER_ID	NUMBER (6)
F		DEPARTMENT_ID	NUMBER (4)
◇		EMP_EMAIL_UK (EMAIL)	
🔑		EMP_EMP_ID_PK (EMPLOYEE_ID)	
<		EMP_DEPT_FK (DEPARTMENT_ID)	
<		EMP_JOB_FK (JOB_ID)	
<		EMP_MANAGER_FK (MANAGER_ID)	
		EMP_DEPARTMENT_IX (DEPARTMENT_ID)	
U		EMP_EMAIL_UK (EMAIL)	
U		EMP_EMP_ID_PK (EMPLOYEE_ID)	
		EMP_JOB_IX (JOB_ID)	
		EMP_MANAGER_IX (MANAGER_ID)	
		EMP_NAME_IX (LAST_NAME, FIRST_NAME)	

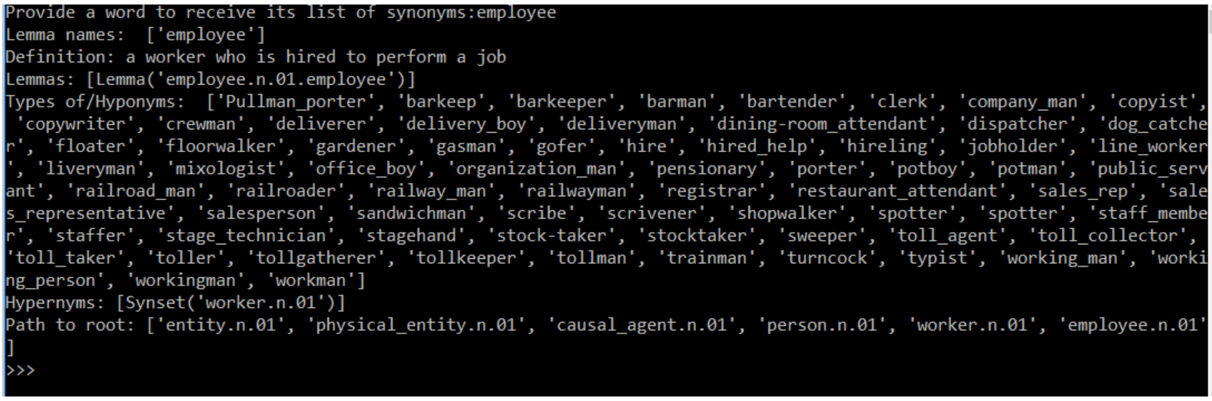
HR.LOCATIONS

P	*	LOCATION_ID	NUMBER (4)
		STREET_ADDRESS	VARCHAR2 (40 BYTE)
		POSTAL_CODE	VARCHAR2 (12 BYTE)
	*	CITY	VARCHAR2 (30 BYTE)
		STATE_PROVINCE	VARCHAR2 (25 BYTE)
F		COUNTRY_ID	CHAR (2 BYTE)
🔑		LOC_ID_PK (LOCATION_ID)	
<		LOC_C_ID_FK (COUNTRY_ID)	
		LOC_CITY_IX (CITY)	
		LOC_COUNTRY_IX (COUNTRY_ID)	
U		LOC_ID_PK (LOCATION_ID)	
		LOC_STATE_PROVINCE_IX (STATE_PROVINCE)	

HR.COUNTRIES

P	*	COUNTRY_ID	CHAR (2 BYTE)
		COUNTRY_NAME	VARCHAR2 (40 BYTE)
F		REGION_ID	NUMBER
🔑		COUNTRY_C_ID_PK (COUNTRY_ID)	
<		COUNTR_REG_FK (REGION_ID)	

Now mapping the employees table to our ontology would immediately identify employee as term related to person. Simple Python script lists above mentioned concepts taken from WordNet.



```
Provide a word to receive its list of synonyms:employee
Lemma names:  ['employee']
Definition: a worker who is hired to perform a job
Lemmas: [Lemma('employee.n.01.employee')]
Types of/Hyponyms:  ['Pullman_porter', 'barkeep', 'barkeeper', 'barman', 'bartender', 'clerk', 'company_man', 'copyist',
 'copywriter', 'crewman', 'deliverer', 'delivery_boy', 'deliveryman', 'dining-room_attendant', 'dispatcher', 'dog_catche
r', 'floater', 'floorwalker', 'gardener', 'gasman', 'gofer', 'hire', 'hired_help', 'hireling', 'jobholder', 'line_worker
', 'liveryman', 'mixologist', 'office_boy', 'organization_man', 'pensionary', 'porter', 'potboy', 'potman', 'public_serv
ant', 'railroad_man', 'railroader', 'railway_man', 'railwayman', 'registrar', 'restaurant_attendant', 'sales_rep', 'sale
s_representative', 'salesperson', 'sandwichman', 'scribe', 'scrivener', 'shopwalker', 'spotter', 'spotter', 'staff_membe
r', 'staffer', 'stage_technician', 'stagehand', 'stock-taker', 'stocktaker', 'sweeper', 'toll_agent', 'toll_collector',
'toll_taker', 'toller', 'tollgatherer', 'tollkeeper', 'tollman', 'trainman', 'turncock', 'typist', 'working_man', 'worki
ng_person', 'workingman', 'workman']
Hypernyms: [Synset('worker.n.01')]
Path to root: ['entity.n.01', 'physical_entity.n.01', 'causal_agent.n.01', 'person.n.01', 'worker.n.01', 'employee.n.01'
]
>>>
```

Repository.

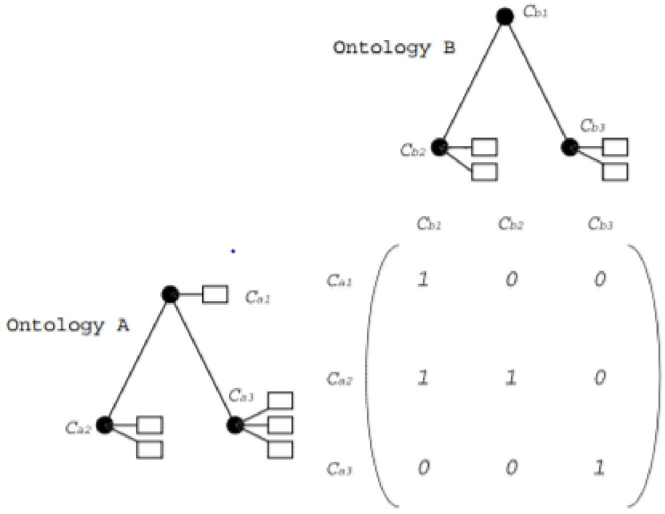
I think that graph databases are particularly well suited to manage both ontologies and metadata. There is a nice write up on how to load WordNet into Neo4J Tom Dickinson here and excellent book by Thomas Frisendal "Graph data modelling".

Automation.

The part of ontology mapping is not entirely clear to me yet but I doubt it is entirely new topic.

One method that I've come across is presented in the paper "Machine Learning Approach for Ontology Mapping".

In that paper the ontology mapping problem consists of defining the value of pairs of concepts in a concept pair matrix, as shown in Figure 2



$$\begin{array}{c|ccc} & C_{b1} & C_{b2} & C_{b3} \\ \hline C_{a1} & 1 & 0 & 0 \\ C_{a2} & 1 & 1 & 0 \\ C_{a3} & 0 & 0 & 1 \end{array}$$

Figure 2. Matrix formulation of the ontology mapping problem.

If two concepts match Ca1 and Cb1 then the value will be 1 else 0.

The authors also suggest a number of similarity measures between two concepts. One of the similarity measures uses WordNet as the knowledge resource for calculating the similarity. It uses synsets to calculate the shortest path of the different word pairs.

The result would be something like this:

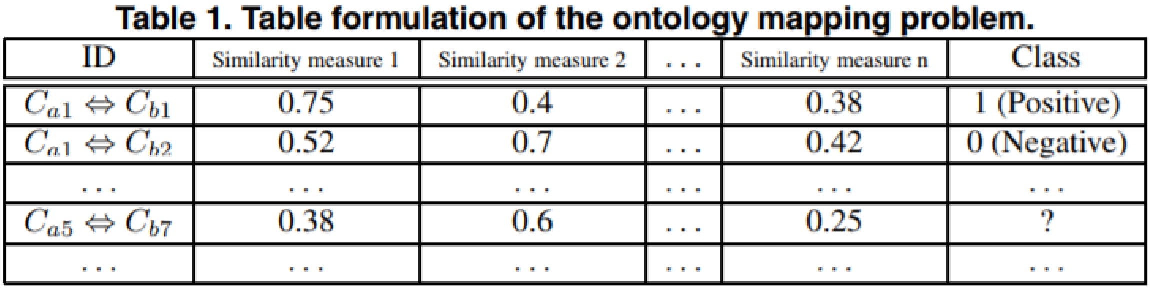


Table 1. Table formulation of the ontology mapping problem.

ID	Similarity measure 1	Similarity measure 2	...	Similarity measure n	Class
$C_{a1} \Leftrightarrow C_{b1}$	0.75	0.4	...	0.38	1 (Positive)
$C_{a1} \Leftrightarrow C_{b2}$	0.52	0.7	...	0.42	0 (Negative)
...	...	...	...	...	...
$C_{a5} \Leftrightarrow C_{b7}$	0.38	0.6	...	0.25	?
...	...	...	...	...	...

Conclusions.

Being able to process humongous amounts of data is great but another thing is to be able to make sense of it. That poses entirely new challenges in front of data modelers. Automating of personal data management would be first step in that direction. Once the necessary infrastructure is established the step to more holistic data management would not be that far.

The gain is obviously tremendous: facilitating and democratizing access to the data, cutting down time spent on preprocessing and making sense of it. The topic of data management is becoming hot. Developing of the well-functioning data library is a natural next step.