Louis Paulet | Dheeraj Poonia

Semantic Knowledge project

Goal : Build an ontology from a text corpus.

Notebook Introduction

All this project was made on a Jupyter Notebook hosted on Google Colab. The notebook will be provided with this report and it contains installation instructions (on how to change the “path” variable to target your local project directory location).
However, it takes a long time to run on the whole dataset (more than 20 minutes) and we recommend to check the link provided below to see a pre-loaded version with all figures and computations already done.

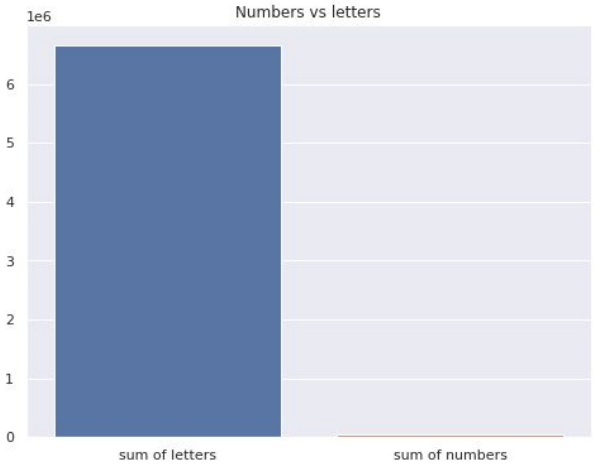
All the plots and metrics displayed in this report are generated thanks to the Matplotlib library (https://matplotlib.org/) and Seaborn (https://seaborn.pydata.org/index.html).

Link to the Ontology building notebook :
https://colab.research.google.com/drive/1ijBdYBuNs1qle8sr2ooCS-_jNSs80NEr?usp=sharing

Step 1 : Analysis of the domain and Core ontology building

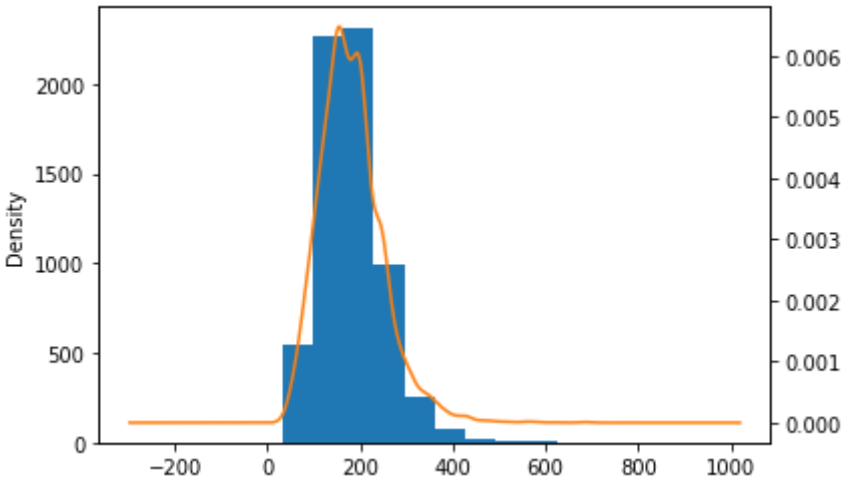
Description of the domain, size, and type

The domain is related to computer science. The size is 6490 lines long, and the type is text.



Very few numbers compared to letters

As a line size is an arbitrary constraint due to data-structure, we can also say that the total corpus size is 1 179 868 words long, with the following repartition :



Word repartition per corpus element

We can see that most word counts are below 200 with some outliers at around 600 words.

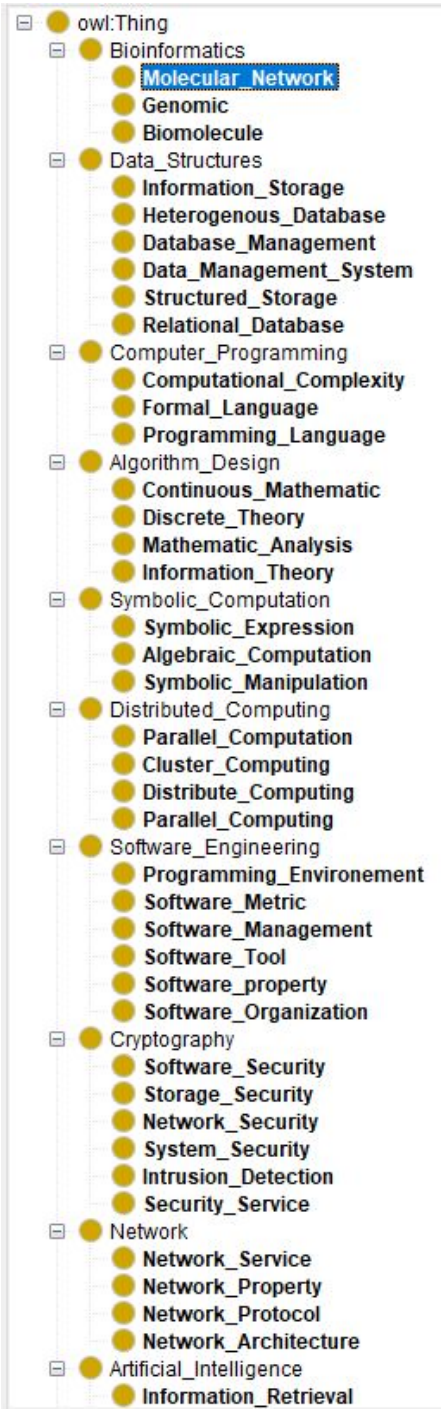
Core concepts

In computer science, these are the core concepts :

- Data structure
- Cryptography
- Software Engineering
- Computer Graphics
- Network Security
- Computer Programming
- Algorithm Design
- Operating Systems
- Distributed Computing
- Machine Learning

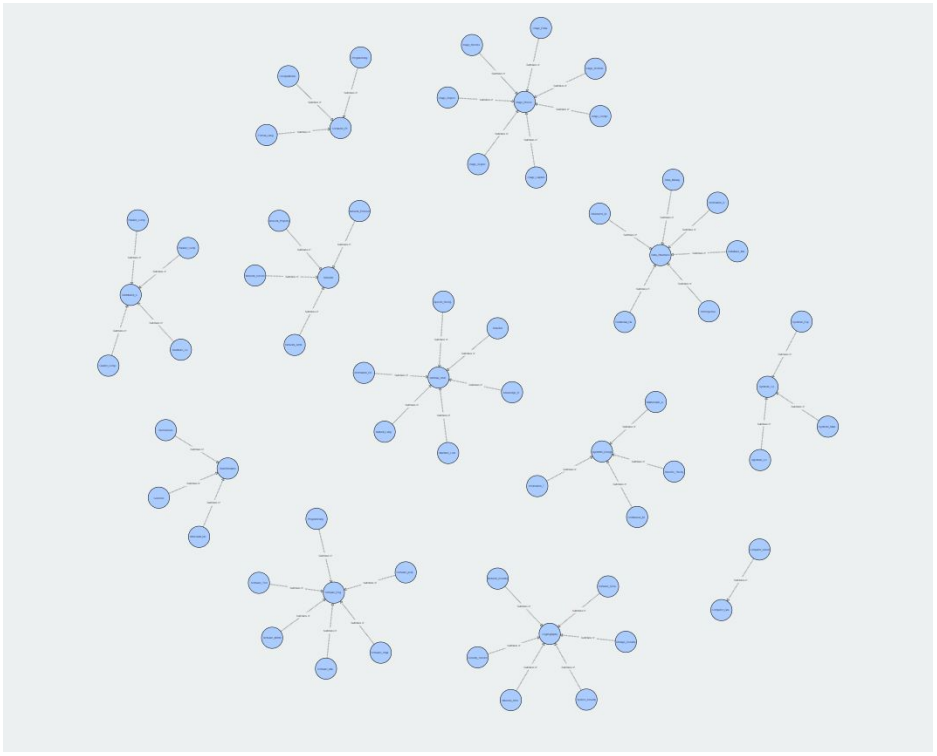
Model of the core ontology

Below is a screenshot of the Ontology classes and subclasses made in protégé :

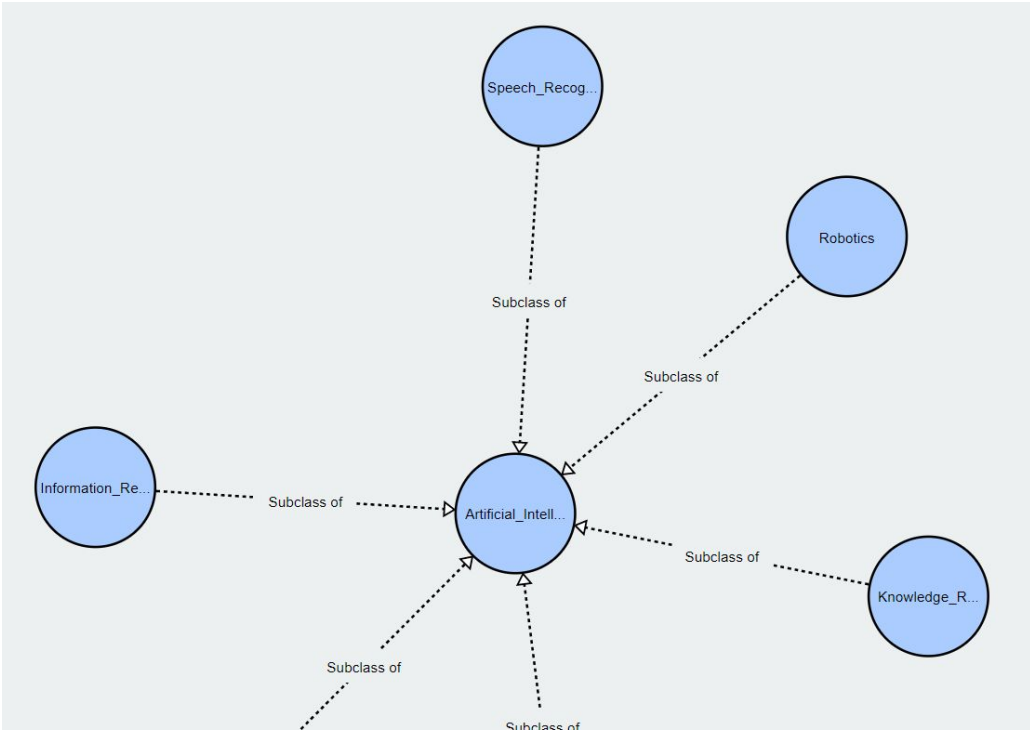


Protégé Class Hierarchy

We can also use this website : http://www.visualdataweb.de/webvowl to visualize the ontology in an interactive fashion. Below are two screenshots to show interesting details, but we recommend to manipulate it on the interactive website by importing the owl file included in this project.



Ontology Global view with a collapse level equal to 0



Zoom on the "Artificial Intelligence" class and some of its subclasses like "Robotics" and "Speech Recognition"

Key terms of the domains

The key terms of the domain are listed in the core-concept section, as well as sub-core-concepts in the provided “TP_CS_CoreConceptIn2Level.csv” file.

Additional terms have been identified in the gold ontology but are more specific and thus, do not qualify as key terms.

These key terms are used as labels in the Gold Standard and Silver Standard datasets.

Step 2 : “Frequent Terms” extraction

Natural Language processing tool selected

We chose to use spaCy as we already tried it in another course. In this section, we only use it to extract sentences and count words. As the frequency of a given word crosses a set threshold (“minFrequency”), we choose to add it to the list of frequent terms.

We did not code this section: the function “corpus parsing” comes directly from the provided files of the ontology building lab work.

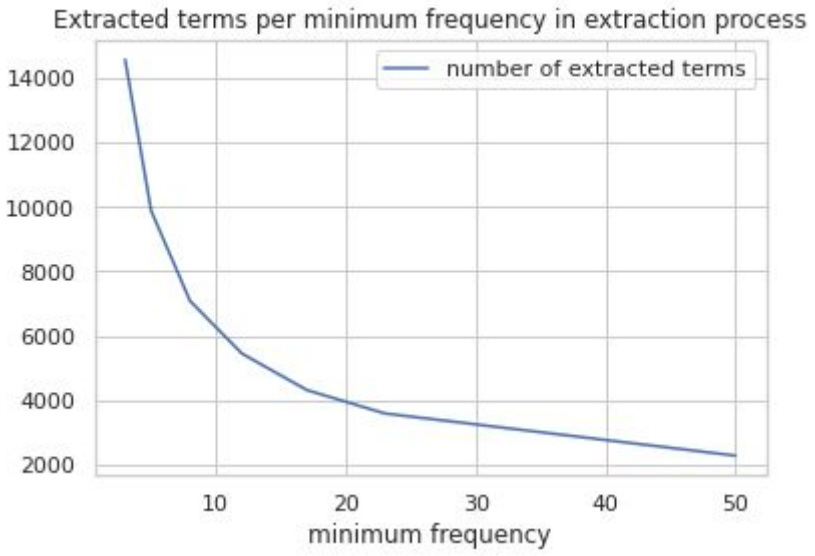
Kind of lexical units to extract

Frequent terms are words that are frequently repeated in the text corpora. These words are mainly defined by the presence of a blank space between them, they can in fact be single characters, or just punctuation.

The objective is to get frequent words which are several letters long, and nouns are the most useful when creating ontologies. However, this step generates a file that is then used in term extraction and these filtering steps are performed later. Thus, we are satisfied with the result from this preprocessing step.

Minimal Frequency selection

Below is a graph obtained from a batch of Frequent Term extractions with different minimal frequencies :



Number of frequent terms per minimal frequency parameter

We can observe that the number of words extracted plummets as the minimal frequency increases. There is an elbow at around minfreq=10 which could suggest that the optimal minimal frequency in regard to execution time could be at 50.
In reality, execution time when executing this function does not increase as minFreq increases, and execution time decreases for the following functions as minFreq increases. Thus it is beneficial to use a high minFreq value in order to have a fast notebook execution.

The limit to this minimal frequency choice is when the number of frequent terms gets below 2000. Clustering and classification algorithms tend to run better on larger datasets, and the co-occurrence matrix used later also needs a large number of terms to apply a high minimal frequency.

We settled on the minFreq=50 value as it is the perfect balance between global execution time and extracted terms results (whose minFreq is also high and needs a large input dataset).

Description of the results of this step

Here is a small sample of the frequent terms :

can
be
describe
by
a
Schrodinger
equation
couple
self
induce

.
for
such
system
,
via
Hirota
method

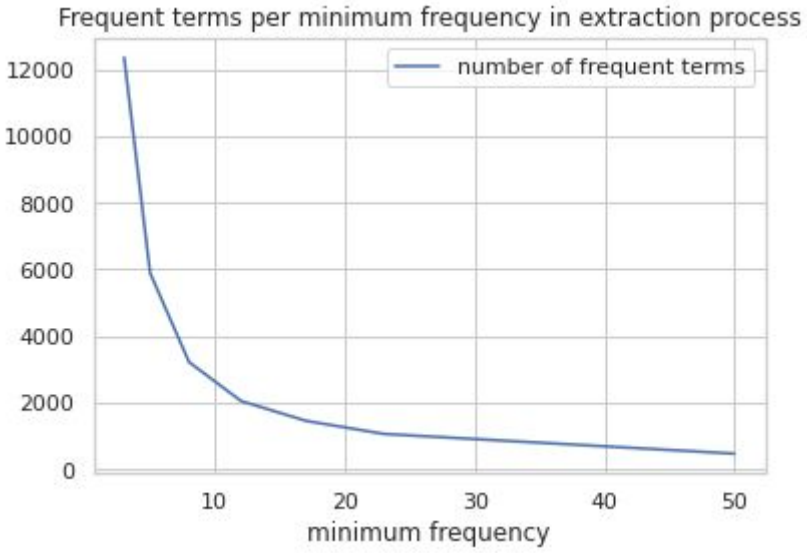
We can see punctuation and stop words. These terms are not useful in an ontology but this is the first preprocessing step.

Step 3 : Natural language processing of the corpus

Description of the step

In the notebook, we call this step “Term Extraction” as it has the frequent terms in input and outputs “real” terms. In the first step, we filter frequent tokens. In this second step, we check whether the extracted token can qualify as a “term”. To build an ontology, we care about certain part-of-speech words, we don’t want single letters or special characters.

The function used in this step is also one of those included in the ontology building lab work (from the file “term extraction.py”). It relies on spaCy to identify STOP words (“that”, “to”, “as”, etc…) and remove them from the frequent terms list. It also applies a second filtering on frequency.



Extracted terms per minimal frequency

We can observe the same dramatically decreasing curve as the minimal frequency increases. Again, we choose a high minFreq value (set at 50) to speed up subsequent steps. The final extraction yields less than 500 words.

Description of the results of this step

Here is a small sample of the term extraction step :

gpu
wsns
reduction
construction
theory
events
years
game
server
survey
engineering
light
sdn
diagnosis
needs
demand

We can see that the terms are actual words that could be included in an ontology on the domain of computer science.

Step 4 : Co-occurrence matrix

The co-occurrence matrix is taken directly from the lab work files. The window size has been tested several times until the matrix as seen in the Matrix Sparsity section of the notebook is as square as possible. We do not notice improved performance with a “long” matrix (high number of columns), only an important size increase.
We do however notice a performance increase in clustering when the square matrix get bigger (as many columns as rows, as many as possible without overflow the RAM of the Google Colab environment).

Step 5 : Gold & Silver standard ontology building

Methodology

The gold standard was built based on frequent term extraction and labeled by hand. The irrelevant terms were deleted, and the relevant ones were labeled with an id. This id

corresponds to the sub-core-concept from the Computer Science sub-core-concepts file given at the beginning of the project (file “TP_CS_CoreConceptIn2Level.csv”).

Due to the term extraction process being agnostic to the Computer Science ontology theme, most of the extracted terms were irrelevant, and thus deleted. Some terms may have been mislabeled as the terms could be attributed to several categories.

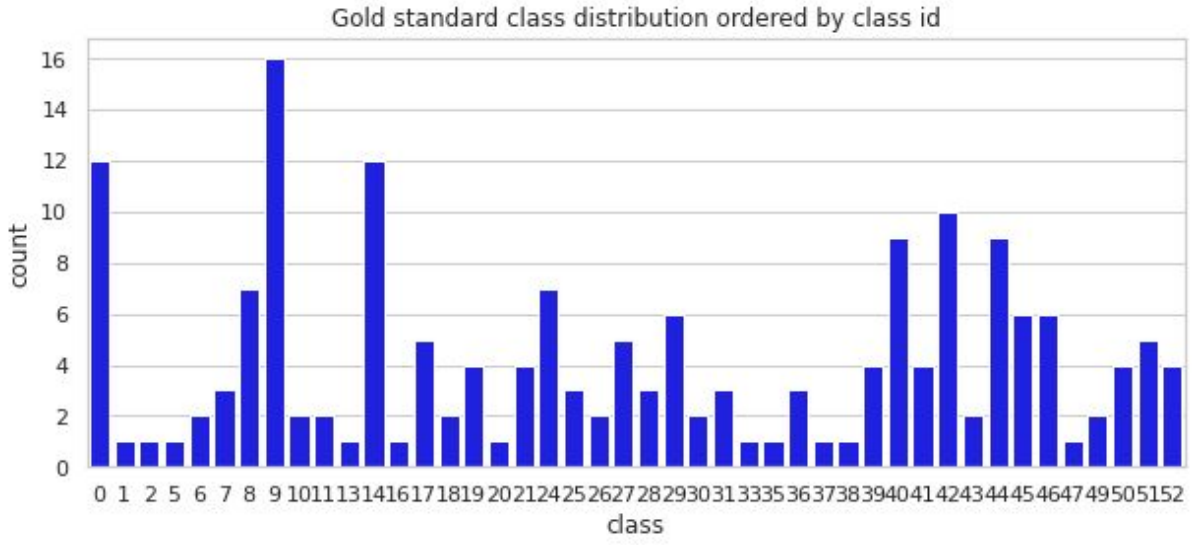
From a list of 600 extracted terms, we ended up with a list of 180 labeled words.

In our project, when we label the extracted terms, the terms that don’t match our Gold Standard have a default label “-1”.

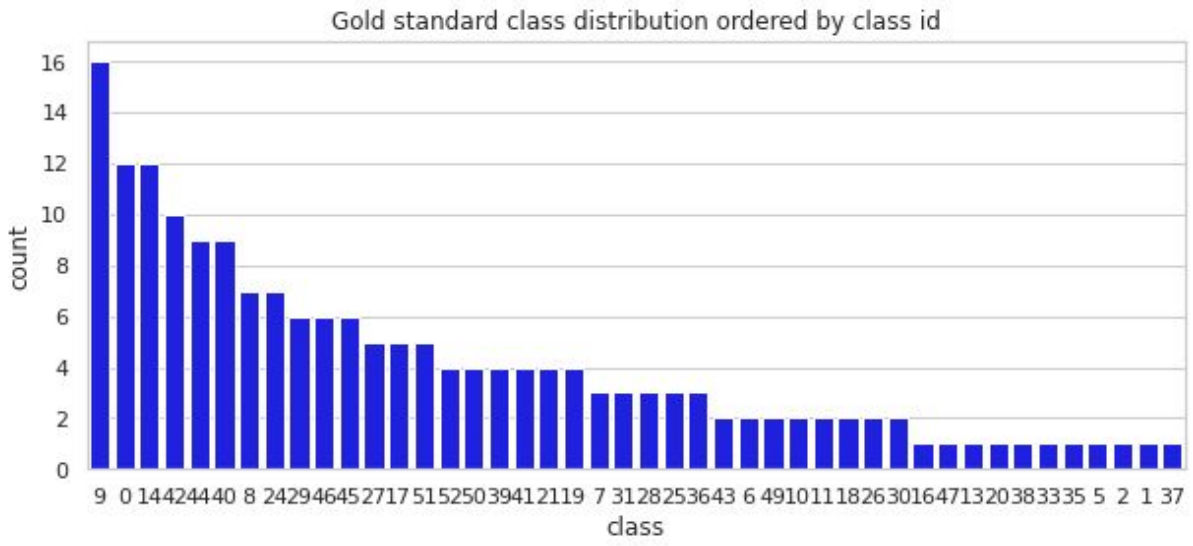
Statistics

By sub-core-categories (Gold standard)

The gold standard is created from a list of frequent terms, therefore, each class is randomly represented. Below is a graph that shows the class distribution ordered by sub-core-category class id to show the high imbalance between classes. Each sub-core-category is associated with a core-concept of the core ontology, we will analyse it in the second part of the statistics (section “By core-concepts”).



Distribution of class counts ordered by class id



Distribution of class counts ordered by class count

If we order the classes by counts, we can see that the distribution follows the rule of Benford law. This means that the text is biased in the subject selection from the start. Most scientific papers must have a certain theme. From the distribution, we can see that the top 3 is composed of class ids 9, 0, and 14 :

9 Artificial intelligence machine learning
0 Computer graphics computer vision
14 network network architecture

Only the first 11 classes are above 5 samples in the gold standard. We will try to use this dataset to train supervised algorithms and report in the results anyway, but we can already find two issues :

- Class imbalance : some classes are 2 to 3 times more frequent than others.
- Class erasure : training an algorithm requires a minimal number of occurrences. Depending on the algorithm, all classes with a low representation will never be predicted.

By core-categories (Silver standard)

Below is a visualisation of the gold standard as seen previously (class count ordered by id value) but with the addition of a hue based on the core-concept class.

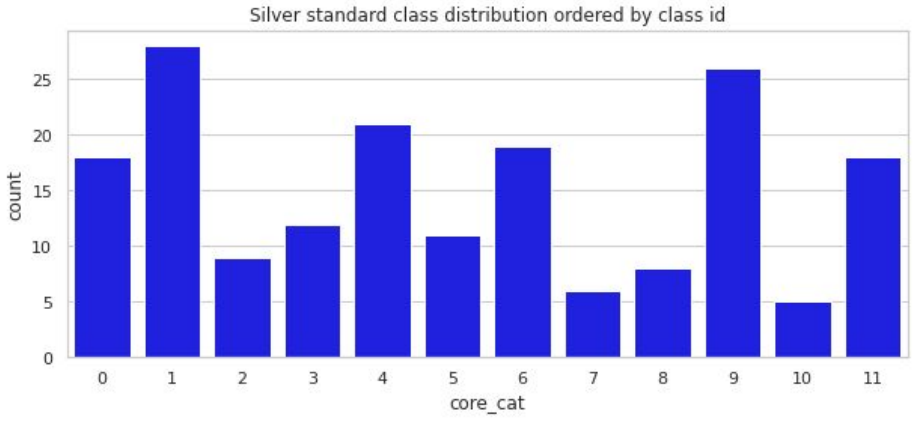
Class count distribution with core-concept hue

As we can see, some core-concepts have more subcategories than others (eg. core-concept 3 has a lot of samples in only one category, core-concept 7 has 5 sub-categories represented, but 3 have only one element, and the last two have 2 and 3 elements). This disparity means that, when training on this data, some sub-categories will never be predicted because they are under-represented, when in fact, the core-category associated has a total count (sum of sub-category samples) higher than other core-categories.

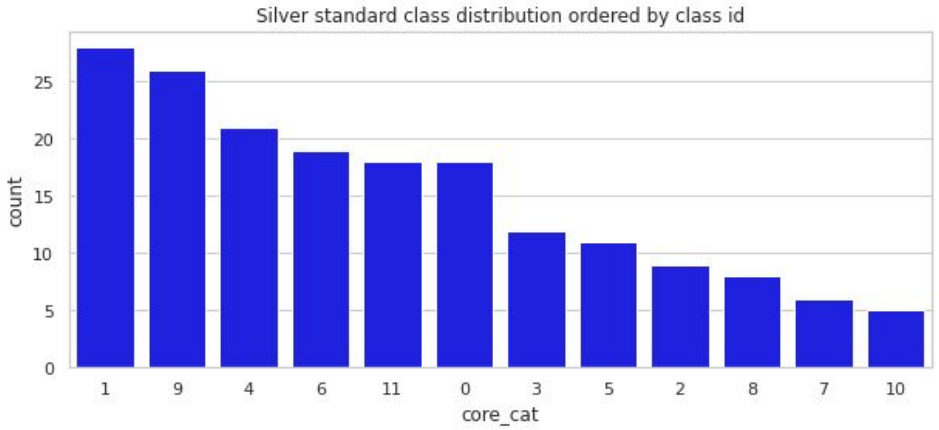
To solve the core-category to sub-category-sum disparities and low class count of some sub-categories, we propose to implement a silver standard.

This standard will be less precise than the gold standard because we cannot infer the sub category from the core-category (we cannot say that a term is part of network infrastructure because we know that it is part of the core-category “network”, but the opposite is possible). However, we hope to get more reliable training with less classes to predict on a small quantity of samples.

Here are the same graphs as the gold standard computed on the silver standard :



Class count repartition ordered by class id



Class count repartition ordered by count

We still observe a class imbalance, with class 1 being 5 times more represented than class 10, but all the classes are represented at least 5 times.
From there, we have 2 choices to correct the dataset :

* we can cut all classes below 10 to ensure a more balanced training but will lose half of the classes
* we can limit all classes to 5 samples in the training set, with the least represented classes suffering from overfitting from the test dataset perspective.

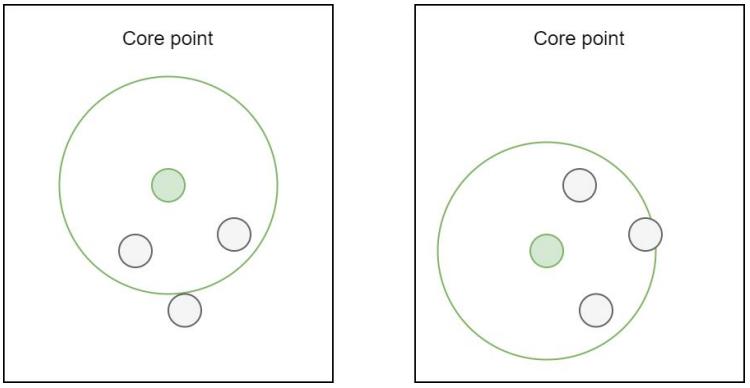
Step 6 : Clustering or classification

Clustering : OPTICS - Unsupervised

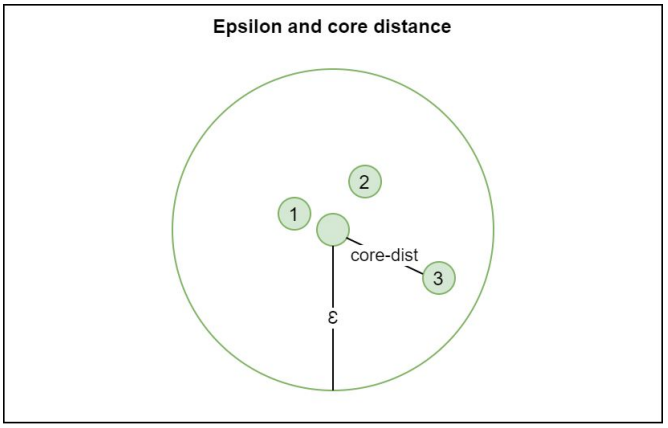
Description of the technique

OPTICS : Ordering Points To Identify the Clustering Structure

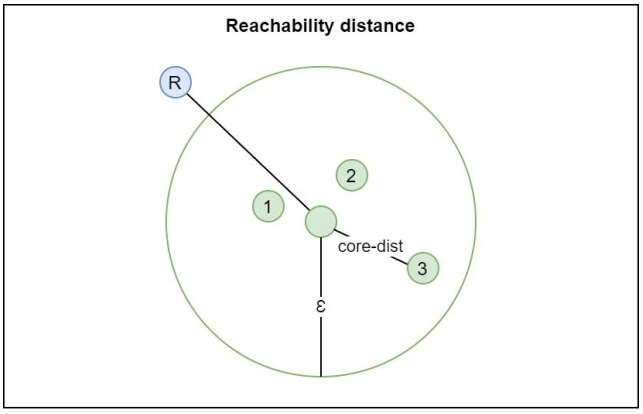
This technique is derived from the more known DBSCAN.
The main idea is to create potential centroids (named “core points”) using parameters and deciding whether the associated points qualify as a cluster.



Each data point gets assigned a value epsilon which will determine how far they can scan around them for data points to add to their list of related points. If a data point is within range, it is added to the list. Its distance, called core-distance, because it is the distance from a random datapoint to the the core, it then used to compute the reachability distance.

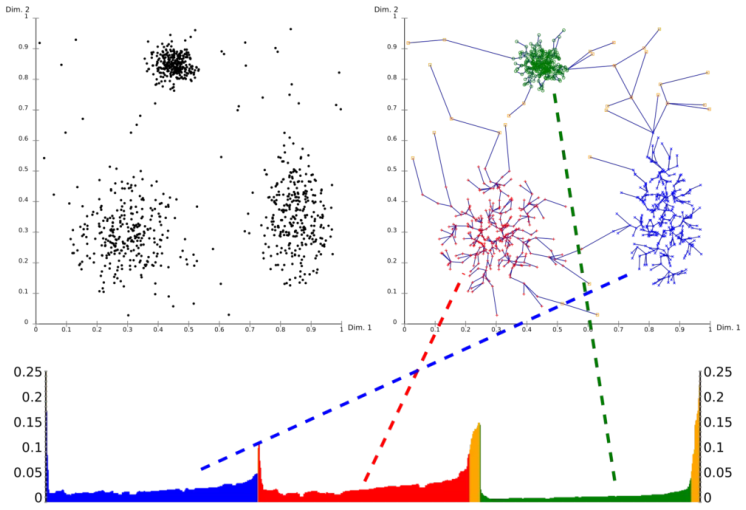


The reachability distance is a parameter whose value depends on the number of points assigned to the core. If the number of points is below a threshold, no need to compute it. If the number of points is above, then we compute this distance and choose to add this new point, or not, depending on the user defined parameters. This technique is used to add more points to the potential future cluster even though the initial epsilon parameters would make it impossible.



Each data point also gets assigned a threshold as a number of datapoints. If they cross this limit, the datapoint becomes the centroid of a cluster. This parameter is called “minPts” and is the threshold that was previously considered in the reachability-distance computation.

The main algorithm principle is to loop through all the data points, assign points to each other depending on their respective reachabilities, and obtain an ordered list of points based on the number of points that can be related to them. When a core gets a high number of points, we can say that its density increases. As we store all these cores in an ordered list, we can select the first N cores from the list based on the user defined “number_of_clusters” parameter in input (if the user wants to find 5 clusters, then the first 5 cores will be selected).



End Result

In the illustration above, we can see the raw cluster (top left), the clusters computed with OPTICS (top right), and the density distribution (based on number of neighbors).

OPTICS paper: https://www.dbs.ifi.lmu.de/Publikationen/Papers/OPTICS.pdf
Illustration source :
https://www.machinecurve.com/index.php/2020/12/15/performing-optics-clustering-with-python-and-scikit-learn/

Implementation

We use the same input files as the kmeans function : “window_matrix.csv”. This matrix contains the co-occurence of the extracted frequent terms, and is fed to the kmeans algorithm, which is also an unsupervised clustering algorithm.

The input is then a NxN matrix, N being the number of extracted frequent terms. This file can go straight to the OPTICS algorithm, no need for any kind of transformation.

The OPTICS algorithm needs user-set parameters (as discussed in the “Description of the technique” section) :

- Epsilon : the maximum range of a core when looking for new points to add in its relation list. It is set to infinity by default, the algorithm can choose the correct value during execution.
- Min_samples : the number of points that need to be related to a core for it to qualify as a new cluster
- Cluster_method : it is set to “xi” (default value). This is the method used to compute distances between points considering steepness (ξ-steep points).
- Metric: which kind of space is considered for the distance computation. The default value here is 'minkowski', which gives a distance computation equal to sum(|x - y|^p)^(1/p), with p being an external variable

To fight against matrix sparseness, we also use PCA (principal component analysis) to reduce the dimensionality of the dataset and have closer points.

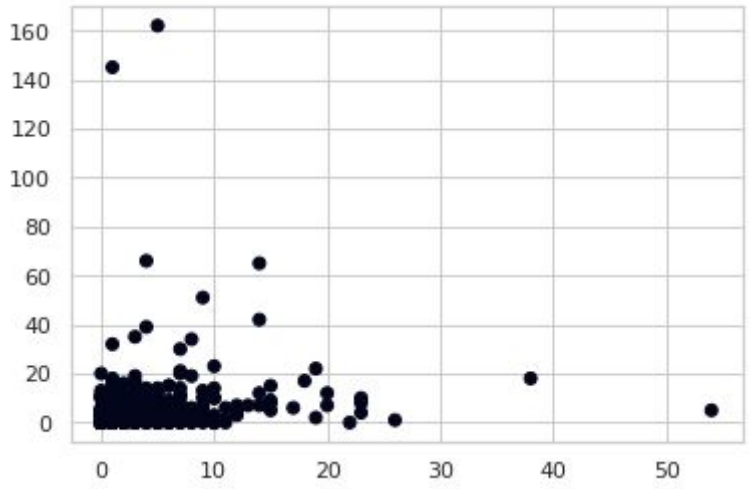
Results of the OPTICS technique

We present two results : one with PCA and one without. To visualize the clusterization, we filter out “-1” values in the output labels. These values represent samples that could not be added to a cluster because their core distance was too high or the clustered points were below the minPoints threshold.

No PCA Result :

```
Unknown Values : 38
Values part of a cluster : 431
How many clusters found : 1
```

Output statistics



One unique cluster identified

Most points are identified. The ones that are clustered are all in the same class. This is a bad result for two reasons :

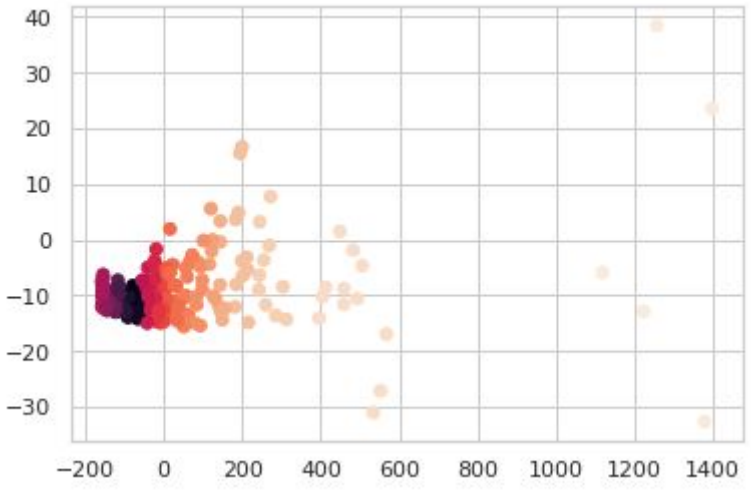
- First, it doesn’t provide any useful information about the identified samples (one big cluster of dissimilar terms)
- Second, the unidentified terms are a strange minority : we don’t know why they were not identified, and why they were far from the other points. The matrix sparseness graph doesn’t show outliers but this result suggests that 38 outliers exist.

It is possible to obtain better results without PCA by decreasing the minPoints parameter (minimal number of points required to form a valid cluster) but the lower the minPoints, the less relevant the classification becomes. When using a 2 point-cluster configuration, we mostly find one term and its plural form (eg. “project” and “projects”) which shows that the proximity between words with the same letters is higher than words the co-occur frequently.

PCA Result :

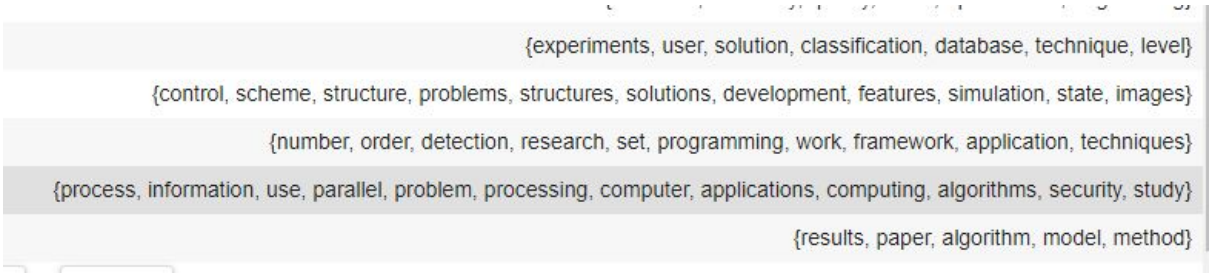
```
Unknown Values : 200
Values part of a cluster : 269
How many clusters found : 27
```

Output statistics



Multiple clusters identified

When using PCA, OPTICS performs much better. The sparseness decreases and correlations can be made. The results are interesting enough to include the results below :



Extract from terms grouped by cluster id

We can see terms that have a different spelling but do co-occur frequently being part of the same cluster.

One big drawback of the OPTICS clustering algorithm is that, as it is unsupervised, we do not obtain the cluster labels. We can see which terms are clustered together but we cannot say that the cluster number 5 is related to network security, or that the cluster number 8 is related to software development.

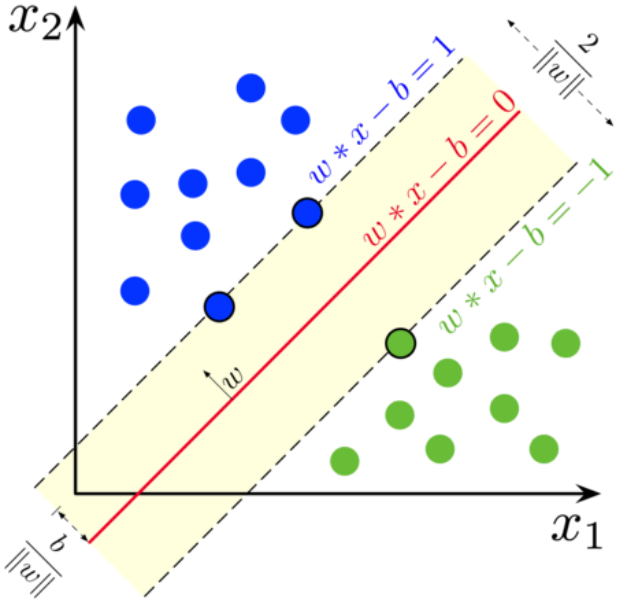
Classification : SVM

Description of the technique

SVM stands for Support Vector Machine. Its goal is to separate a dataset into two classes. Therefore, it can only be used for binary classification (and regression). To use it in this multi-class context (building an ontology with multiple core-concepts), we will have to add a few modifications around the algorithm.

SVM works by computing an hyperplane that linearly separates data into two classes. The hyperplane is just a line, and separating two classes by a line would be dangerous as very minor differences in data point properties would make the data points close to the line change class. This is why the hyperplane has a margin.

No data points are allowed to be inside the hyperplane margin. In fact, the points that lie at the limit of this margin are called support vectors :



In red, the hyperplane. In yellow : the margin.

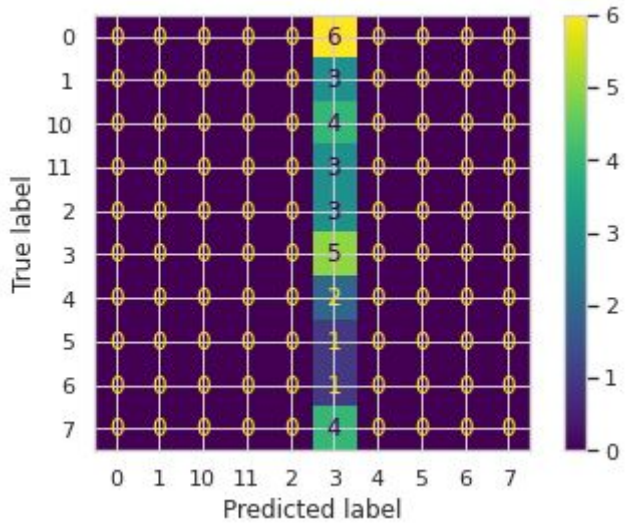
As we can see on the above graph, the equation for the margin is determined by these support vectors. The blue class has two data points that, when placed on a margin, provide the vector of the upper margin. By placing parallel lines at an equal distance, we can obtain the hyperplane and lower margin. The red line is the hyperplane that classifies the data into two classes : the blue class and the green class. From the above illustration, we can also say that the most accurate (in terms of precision and recall) SVMs have the widest margins.

The above example is simplified as it has to be shown in 2 dimensions, but this concept works in as many dimensions as needed. The hyperplane doesn't have to be linear, it can also be expressed by a polynomial function (by switching from linear support vectors to polynomial support vectors).

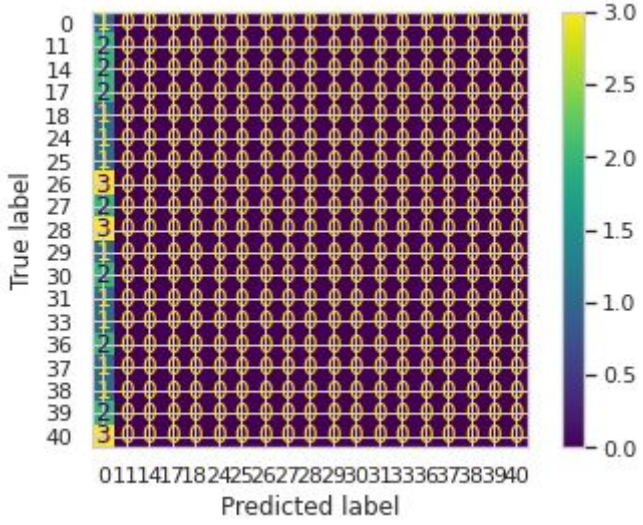
Implementation

While implementing the SVM method for the classification. We took a training/test split as the most commonly used ratio of 30:70. Using the Sci-kit learn library we used the SVC function for fitting the data and performed for both gold and silver ontology.

Results of the SVM

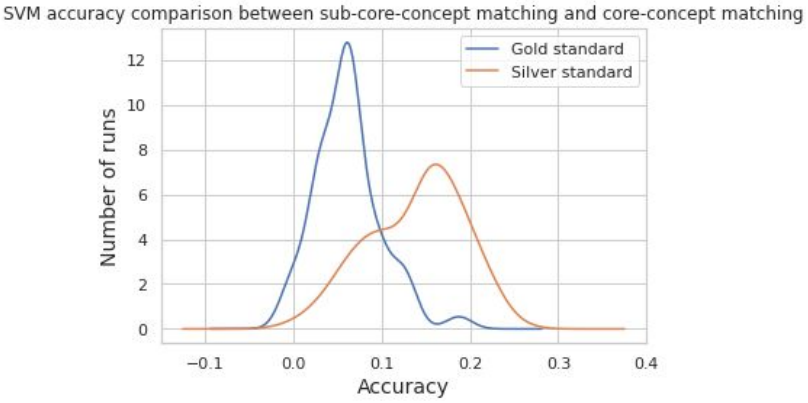


Silver standard confusion matrix

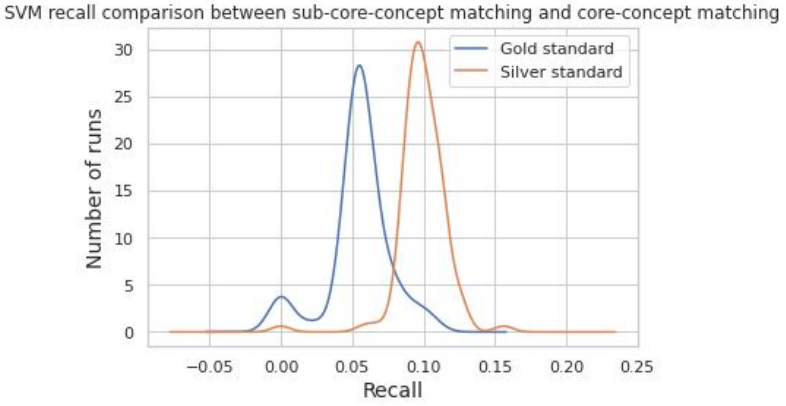


Gold standard confusion matrix

For both of the ontologies the classification algorithm only gives one class or core ontology which can't be considered as a good result. It is due to the less identification of the terms and due to which there are not many terms to train for data also to get a good accuracy.



Accuracy per training session in SVM on gold standard vs silver standard



Recall per training session in SVM on gold standard vs silver standard

Conclusion

We have extracted frequent terms and interesting terms using NLP. We used those terms to build a co-occurrence matrix and build a gold standard. From this basis, we used clustering and supervised methods to automatically create an ontology.

The supervised results are bad. The accuracy is poor by default, and when lowering our standards (by the use of a "silver" standard), we get around 18% accuracy, which is still too low. As we have seen in the Gold and Silver standard analysis section, this is due to a small dataset with low class frequencies and an imbalance in the class repartition. A better technique would have been to either crowdsource a better gold standard with more words linked to more core concepts, or use an existing ontology and use it as a training dataset.*

The unsupervised model (OPTICS) has better results as the dataset size is only limited by the respective frequencies of the word extraction and term extraction steps, as well as the size of the co-occurrence matrix algorithm. No manual step in this process came as a bottleneck to hinder the model performance. In this case, we solved the problem of high distances by dimension reduction using PCA. Before PCA, no clusters were formed because

the detected terms were too dissimilar to be considered close to a core data point. With PCA, the Minkowski distance has been virtually reduced and multiple clusters can be formed without lowering the minimal number of points needed to qualify for a new cluster. The only drawback to this method is that, although words are linked together, there is no way to label the clusters relative to the Gold standard (in terms of core-concepts).

To conclude, this project shows that building an ontology automatically from scratch is a difficult process. It is always better to start with an existing ontology and build from there. We couldn't choose the algorithms to use as they were allocated to us during the project kickstart phase, which slowed us down on the OPTICS model that we had to learn how to use the parameters and the influence of core distances in the results. On the supervised model, a random forest would have performed better with a small dataset than an SVM.