NTNU
Norwegian University of
Science and Technology

A Framework for Ontology Based Semantic Search

Thomas Fjæstad Hagelien

Master of Science in Informatics
Submission date: July 2018
Supervisor: Jingyue Li, IDI
Co-supervisor: Shanshan Jiang, SINTEF Digital

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Publicly-accessible open transport data is provided by the public sector in an effort to create new opportunities, stimulate innovation and enable new solutions that benefits the society. The number of datasets available are however limited. This is partially due to the necessary, but labor intensive, preparation process of each dataset. The datasets need to be annotated with descriptions that explain their purpose and content. The search and retrieval functionality of current publishing platforms are limited to classical keyword based search, which is much more restricted than the search technology used for finding information on the world wide web. This is due to the fact that information in most cases cannot be retrieved directly from the data itself, but depends on the dataset descriptions. Open Datasets are encoded in a rich variety of formats which makes it difficult to reuse them directly in software applications. This study investigates how a transport domain knowledge model, namely an ontology of the transport domain, can enable data to be identified in terms of its meaning in a given context, i.e. semantics, and not by keywords and tags alone. The study further to investigates how semantic technology can be applied to improve discoverability and reuse of datasets. This was done by initially developing a prototype framework for ontology based semantic classification. The framework works as a test bed that allows for different algorithms to be tested and compared against different ontologies. The framework also includes the development of an online search engine that is used to measure the efficiency of the data discovery method. This study further includes a conceptual design for a software system that allows transport related software applications to utilize datasets from heterogenous sources. The study finds that automated classification based on natural language processing of dataset descriptions is possible and shows promising results. This approach appears to improve the search and retrieval functionality of limited datasets, however it is currently sensitive to the quality of the description text and needs to developed further.

Thesis Proposal (Problem Description)

Open data fuels the development of new and innovative ICT solutions for the transport sector. Better catalogue services for discovery of open data are needed to support the use of such data and information in new applications and services that make transport sector more efficient, secure, safe and green. There are catalogues on organizational level (e.g., for individual transport authorities), on local level (e.g., for a municipality), and on national level. Catalogue services for Norwegian open data are currently provided by DIFI (http://data.norge.no). Norwegian Public Road Administration (Statens vegvesen) is planning a portal based on the ckan.org open source. These catalogues need to be federated (i.e. connected) to support discovery of all open data.

The project will design and develop a prototype federated catalogue system for semantic discovery of open transport data and data characteristics (e.g., security issues, quality, type of data) across different data sources and providers. Semantic technologies based on for example metadata, ontologies and open linked data (LOD) will be utilized to enable applications to find relevant data during runtime, also data depending on the location.

The tasks include literature study, prototyping, evaluation and documentation. Literature study includes review of available technologies for semantic search and for representations of metadata.

The prototype will include functionality for automatic or semi-automatic semantic annotation, and semantic search combining search based on ontology and metadata. The prototype will be built based on the CKAN open source platform and use catalogues / datasets from DIFI and Statens vegvesen as the starting point.

The project will be suitable for one student. The software will be open source.

This proposal relates to the NFR project "Open Transport Data", where Statens vegvesen, SINTEF, ITS Norge, Kystverket and Oslo kommune are partners.

Preface

This master project was suggested by Dr.Ing. Shanshan Jiang (Research Scientist at SINTEF Digital), acting as co-supervisor, and defined together with my supervisor Assoc. Professor Jingyue Li at the Department of Computer and Information Science (IDI), NTNU. The project addresses one of the focus areas for the NFR financed Open Transport Data (OTD) project at SINTEF Digital. The project work was carried out during autumn semester of 2017/spring 2018.

I like to thank my supervisor Assoc. Prof. Jingyue Li for your support and guidance, and my co-supervisor Dr. Shanshan Jiang for your good feedback and discussions. Also a big thank you to Dr Marit Kjøsnes Natvig for the contributions on the ontology development and manual tagging of datasets. A huge thank you goes my dearest Nadra J. Nilsen for your inspiration, support and patience, and for giving me the time to work on this project. Without you I would not be able to complete this.

Trondheim, 2018-06-01

Thomas Fjæstad Hagelien

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Open transportation data is a type of government data that deals with transportation services, infrastructure, rules and regulations, statistics and other types of information. This section gives a brief introduction to the history of open data and current initiatives.

A Brief History of Open Data

The idea of making government information transparent and accessible was initiated in May 9th 2013 by the Obama Administration. In an Executive Order the Open Data Policy was released to "Enhance Government Efficiency and Fuel Economic Growth".[1]

The memorandum states that the term "*open data*" refers to publicly available data structured in a way that enables the data to be fully discoverable and usable by end users. In general, open data will be consistent with the following principles:

- Public Agencies must adopt a presumption in favor of openness to the extent permitted by law and subject to privacy, confidentiality, security, or other valid restrictions.
- Accessible Open data are to be made available in convenient, modifiable, and open formats that can be retrieved, downloaded, indexed, and searched. Formats should be machine-readable (i.e., data are reasonably structured to allow automated processing). Open data structures should not discriminate against any person or group of persons and should be made available to the widest range of users for the widest range of purposes, often by providing the data in multiple formats for consumption. To the extent permitted by law, these formats should be non-proprietary, publicly available, and no restrictions should be placed upon their use.
- Described Open data are to be described fully so that consumers of the data have sufficient information to understand their strengths, weaknesses, analytical limitations, security requirements, as well as how to process them. This involves the use of robust, granular metadata (i.e., fields or elements that describe data), thorough documentation of data elements, data dictionaries, and, if applicable, additional descriptions of the purpose of the collection, the population of interest, the characteristics of the sample, and the method of data collection.

- Reusable Open data are to be made available under an open license that places no restrictions on their use.
- Complete Open data are to be published in primary forms (i.e., as collected at the source), with the finest possible level of granularity that is practicable and permitted by law and other requirements. Derived or aggregate open data should also be published but must reference the primary data .
- Timely Open data are to be made available as quickly as necessary to preserve the value of the data. Frequency of release should account for key audiences and downstream needs.
- Managed Post-Release A point of contact must be designated to assist with data use and to respond to complaints about adherence to these open data requirements.

The Open Transport Data project

The background for this thesis is the Open Transport Data project (OTD project). The OTD project is funded by the Research Council of Norway since 2016. The project will end on April 2019. SINTEF Digital with Dr. Marit K. Natvig, is leading the project. SINTEF Digital also co-supervises this thesis with Dr. Shanshan Jiang. The OTD project owner is the Norwegian Public Road Administration. Other project partners are:

- Norwegian Coastal Administration
- Municipality of Oslo
- ITS Norway
- URBALURBA

The project aims to develop a virtual lab offering that facilitates data sharing in order to utilize open transport data efficiently. The overall idea of the OTD project is illustrated in Figure 1. (This figure is taken from [2]. More information about the OTD project can be found on the project website [2].

The public sector produces (and/or finances) a vast amount of data. This data is often referred to as Public Sector Information (PSI) and may be anything from weather related data, geographic information and statistics, to open access data from research projects. The data that is granted under non-restrictive condition is called Open Data. Open Data allows for anyone to access, use and share data between governments, businesses and individuals. For many innovative companies, Open Data is a key asset. Governments promote open data for both transparency and economic growth. The EU open data market is an example of the economic impact of publicly founded data. The European Commission has policies to re-use Open Data for economic and societal benefits. In the transport sector, there are EU directives stating that transport data must be publicly available. The motivation for

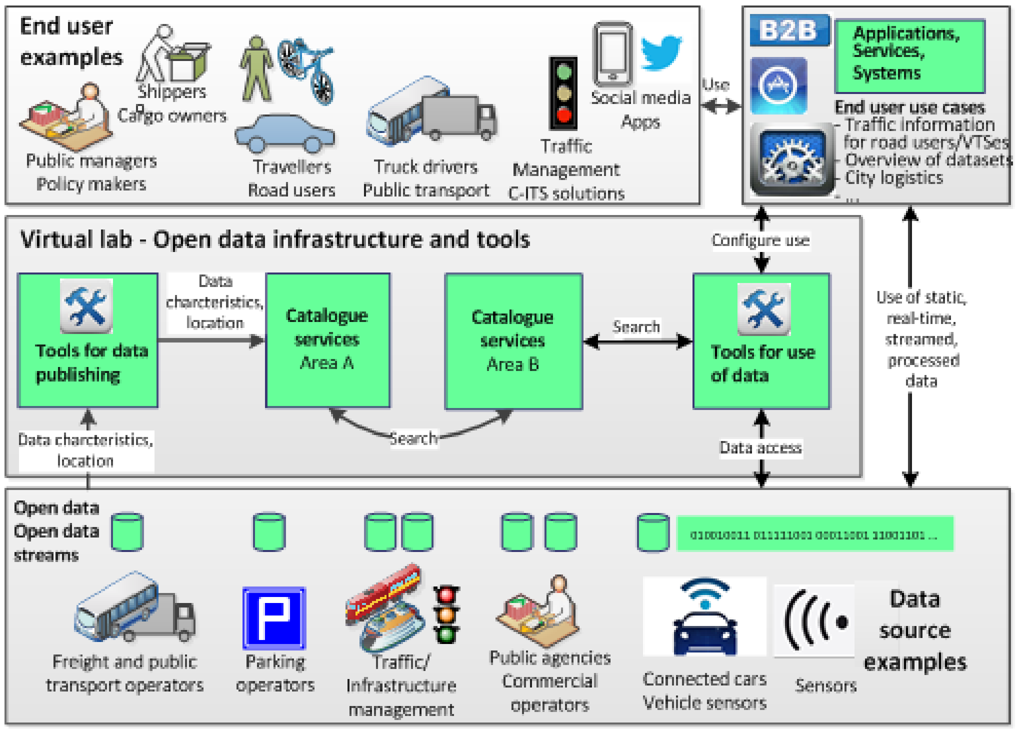


Figure 1: The Overall idea of the Open Transport Data project

this is to stimulate innovation, that produces digital solutions.

In this project the focus is on *transport data*, although data for different sectors may share the same properties. There has been a significant growth in the amount of transport data produced, but datasets published as open data are still quite limited. One contributing factor to this is the lack of existing infrastructures and solutions to facilitate data sharing. First of all, the process of making a dataset open is typically very resource de manding since rigorous quality control, documentation and correct formatting are required in order to produce a dataset that meets user expectations. Once the datasets are published, the questions is how users can find the most relevant datasets. Search technology is an active research area that has undergone a paradigm shift in later years. While it was previously necessary to construct a search query using a strict syntax, it is now possible to formulate a search query using natural language. This has greatly increased the usability of web-search engines. Information on websites is however stored in a structured and well defined text format which simplifies data retrieval from these sites. In contrast, open data are encoded in a wide variety of formats which greatly complicates search, reuse and retrieval of these data. The datasets need to be annotated with descriptions that explain their purpose and content and the search and retrieval is restricted to keyword based search. This greatly limits the retrieval and reuse of these data and is a major obstacle in utilizing open data.

1.2 Project Goals

The goal if this project is to study how information retrieval and data reuse can be improved for open data in general, and open transport in particular. The hypothesis is that ontology based semantics, i.e. connecting open transport data with a model for transport domain knowledge, leads to significant improvements in information organization, dataset search, retrieval and reuse. The project aims to produce a test bed for ontology based semantic classification and search through which technologies can be tested and ontologies be improved.

Open Data versus the Web

Semantic search engines are now very common. The most popular search engines allow users to type sentences in any human written language, and results are for the most part highly relevant. So why should it be any different for published datasets? Search engine companies use bots to automatically collect web pages. Web pages are stored in a structured format called Hypertext Markup Language (HTML). In a process called web scraping, text is extracted from the HTML pages. The text is further analyzed to identify the context and specific properties. This information is then linked to other sources of knowledge in a large graph. Information is extracted from the graph based on the context of the search query.

Open Data differs from web documents in numerous ways. Perhaps the most important difference is that there is no common format for which the data is published. Search engines cannot easily extract any information from the published datasets, without having some extra knowledge about what this data is. To illustrate the importance of adding contextual information to data, consider a dataset containing the following list of numbers: [13,11,16,17]. What is this? It is completely meaningless unless we know that these values represents average temperatures in 6 hour intervals. If we add a location and a date, we know even more. The full picture is not reveled until we add that the temperatures are given in Celsius degrees. This might be obvious to a human, but not necessarily to a machine. To make matters worse, the list [13,11,16,17] can be formatted in numerous ways and published as proprietary text or binary formats. The data itself may furthermore be unreadable unless you have purchased a license to the software that is able to read it. Thus, Open Data differs very much from web contents, and requires careful preparation in order for it to be useful.

Research Challenges

In the following, the term *ontology* will be used for the formal knowledge description of a domain, while the term *taxonomy* will be used to describe a classification tree of concepts. In literature the terms are often used interchangeably, but here *ontology* is applied to describe the entire knowledge domain, while taxonomy to specifically address the classification.

Challenge 1 - Representative Transport Ontologies

In the domain of transportation there already exist ontologies. These ontologies are defined for specific purposes, like describing concepts related to public transportation route planning. While the available open transport datasets are limited in numbers, their variance in context reaches far beyond existing ontologies. Due to this variance, it is therefore difficult to make use of them since this complicates search and retrival.

Challenge 2 - Search

Current distribution platforms of open datasets such as CKAN are limited to simple keyword search. The keywords are extracted from the available metadata. Search is therefore limited to the language used to defined the metadata. This has a clear disadvantage in federated catalogue system where datasets can be described using numerous languages. By issuing a keyword in one language, relevant datasets are left out because they are annotated using another language.

Challenge 3 - Information exchange

As previously mentioned, the dataset may reside in different data formats and/or services. This is especially challenging for software applications that makes use of the information. In Electronic Government the standardization efforts for information sharing and interoperability is limited. The lack of mature semantic interoperability frameworks and tools limits an effective utilization of this information[3].

1.3 Research Questions

In order to address the challenges identified in the previous section, the following research questions were formulated;

RQ.1 How can the design of transport ontologies be improved to support the classification of current and future dataset?

- How can transportation domain knowledge be translated into an ontology

RQ.2 Can semantic technology be applied to improve the discoverability of open transportation datasets?

- How can language independence be supported? (i.e. how can search be made independent of the language used to describe the datasets).
- Can an automatic- or semi-automatic classification process be developed to replace manual tagging?

RQ.3 How can domain applications effectively utilize a wide range of available open datasets published in different formats?

1.4 Project Description and Approach

This project builds on the Open Transport Data project (introduced in 1.1). The project has 3 key modules. An overview of the project shown in Figure 2. The published datasets are the key assets, and we examine how these could be prepared to increase their reuse value. The project aims to develop a proof-of-concept ontology design for testing, and provide future recommendations for the transport domain.

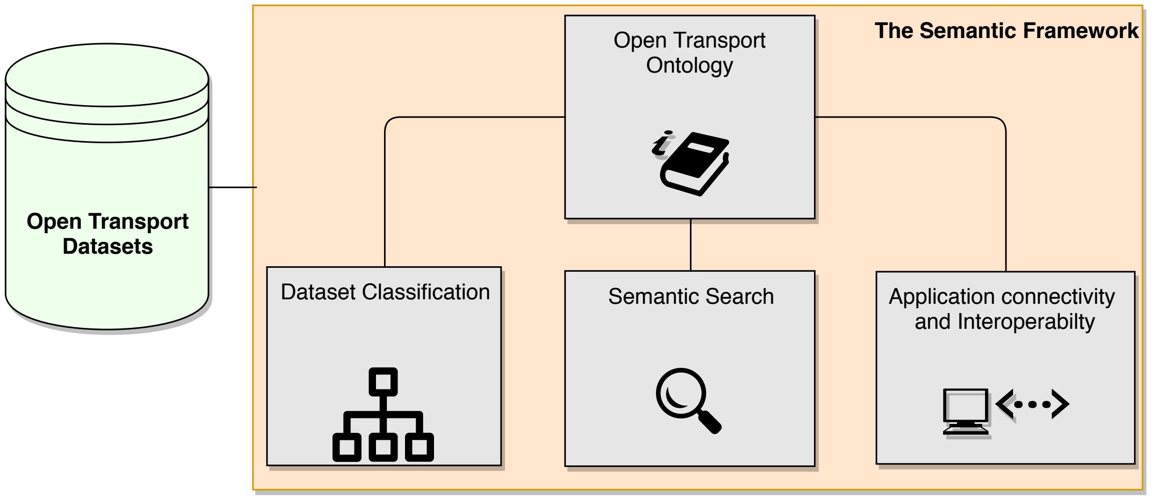


Figure 2: An overview of the ontology based semantic framework developed in this project. The framework is a testbed for studying classification, search and interoperability.

Formal domain knowledge facilitates communication between different features for the semantic framework, as we demonstrate in 3 applications. The first application is a dataset classification application that uses domain knowledge to organize information. The second application is semantic search, which use natural language processing in combination with the classification information to identify relevant datasets. The third application is an interoperability service allows a domain application to utilize data from datasets without having to know about its data format.

Design Science Research Methodology

The project focuses on improving an existing functionality, i.e. search and utilization of open transport data. The research methodology chosen was the **Design Science Methodology**[4] where the objective is to improve the functionality of an artifact (in this a software system to perform search and facilitate reusability) through several process iterations of design, implementation and evaluations improve on the solution.

1.5 Project Contributions

Design of a transportation domain knowledge model

This project provides insight into the design of a knowledge representation (ontology) of the transportation domain, and suggests a prototype for further development. The transportation ontology is modelled in cooperation with domain experts. Open transportation datasets are associated with relevant concepts. This defines a semantic model of the open data catalogue.

A prototype framework for ontology based semantic search

A prototype framework for ontology-based semantic classification and search, allows for ideas to be conceptualized, tested and evaluated.

A method for semantic classification and tagging of transportation datasets

A technique for automatic classification and tagging of datasets has been developed. The method combines semantic similarity measurements from lexical databases with measures of semantic similarity between concepts in the ontology.

Evaluation benchmark

A set of Norwegian transport data have been manually tagged and prepared as an evaluation benchmark. This benchmark can be reused by succeeding projects to measure improvements.

Language independent semantic search

By allowing the search-language to be independent of the language used to describe the different datasets, allows for a unified search interface for a multilingual federated data catalogue service.

Semantic interoperability of domain applications

A design for allowing domain applications to work with heterogenous datasets from multiple sources is presented. Based on previous work by the author, a method is examined that permits a software application to discover and acquire data from heterogeneous data sources.

1.6 Thesis Outline

Background and introduction

The initial sections of Chapter 1 gives a brief introduction to Open Data in general, and an introduction to the Open Transport Project.

State of the art

The state of the art (Chapter 2) discusses current open data catalogues and publishing platforms (Chapter 2.1), automatic dataset classification (Chapter 2.2), metrics for semantic similarity (Chapter 2.3) and semantic interoperability (Chapter 2.4).

Development

The research questions are addressed in the three chapters; Ontology Development (RQ1) (Chapter 3), Tagging and Classification (RQ2) (Chapter 4) and Semantic Interoperability (RQ3) (Chapter 5) The semantic framework, which was built as part of the thesis for the purpose of testing different techniques and evaluating its results, is presented in Chapter 6.

Evaluation and Conclusions

The evaluation is presented in Chapter 7 followed by a Discussion and Conclusion (Chapter 8).

2 Technologies and State of the Art

This chapter describes the current state of the art relevant to the research questions. The main topics cover the availability and discoverability of open data in current publishing platforms, semantic technology used for classification and interoperability.

2.1 Open Data Catalogues and Publishing Platforms

There are a number of open data publishing platforms aimed at governments, organizations and companies. The Comprehensive Knowledge Archive Network (CKAN) is an open source solution developed by the Open Knowledge Foundation[1]. CKAN is designed to be modifiable and highly customizable. The back-end of the CKAN web server is implemented in the Python programming language[5], dataset information (metadata) is stored using PostgreSQL databases[6]. CKAN also comes with a search engine implemented using Apache Solr. Solr is an open source enterprise search plaform[7]. Solr supports full-text search, and has support for different ranking functions like BM25[8] and TD-IDF (see chapter 4.1. In addition, Solr can be extended with a Learning To Rank (LTR) module. which is based on machine learning models, and improves the ranking of the retrieved documents.[9]For examples of CKAN usage, see chapter 7.2.5.

The EU Open Data Portal is central to the Digital Single Market (DSM) and open data strategy. The portal is developed using different Open Source Platforms, including CKAN. Users may search data organized in a metadata catalogue. Other Similar platforms as CKAN are Drupal[2],DKAN[3] and Socrata[4]. These solutions do not come with their own search engines, but rather relies on external search engines to crawl the index of published pages.

[1]CKAN system for publishing and managing datasets (i.e. collections of data). Typical users of CKAN includes governments (local and national) and research organization.

[2]Drupal claims to be leadning open-source content management system (https://www.drupal.org/ (Accessed 2018-07-01)

[3]The DCAN Open Data Platform is free and open source platform for government open data https://getdkan.org/,(Accessed 2018-07-01)

[4]Socrata is a commercial, fully supported (turn-key) Software-as-a-Service solution https://socrata.com/

2.2 Ontology-Based Classification

A classification system is an system where "things" are arranged based on context, similarities or other logical relations. Ontology-Based Classification use domain knowledge as the context for organizing information. The classification process assumes some knowledge about what is to be classified and how this relates to the domain. A human domain expert may be able to accurately identify the appropriate context of a specific piece of information, but the process might be very time consuming. By using computers, the classification process can be supported by software that makes the process more effective. Automatic classification techniques are active research topics, and often studied in conjunction with emerging machine learning algorithms.

There are four general types of classification types:

- **manual classification** is the process of having a human domain expert performing the classification process
- **supervised classification** is a method that is based on having a set of training data (or human feedback).

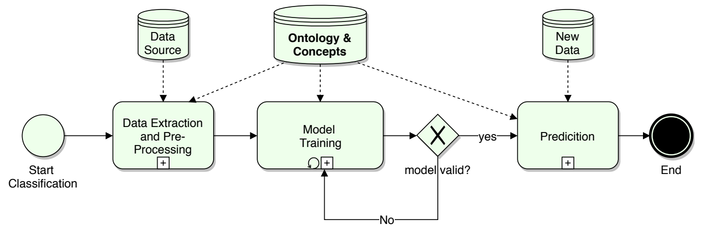


Figure 3: Supervised classification workflow

  Figure 3 illustrates the basic workflow of the supervised classification method. A function that takes a dataset and/or dataset description and suggests a category is called a *mapping function*. The mapping function is improved by making predictions on the training data and corrected based on the accuracy of the prediction. The improvement step is repeated until the mapping function predictions are acceptable.
- **unsupervised classification** produces a mapping function based on patterns or structures found in the dataset or dataset descriptions.
  Figure 4 illustrates the basic workflow of the unsupervised classification method. The main difference from the supervised classification method is that there are no iterative steps that adjusts the model based on test and training data.
- **semi-supervised classification** is where only a subset of the datasets are categorized (unlabeled). A mix of supervised and unsupervised learning tech-

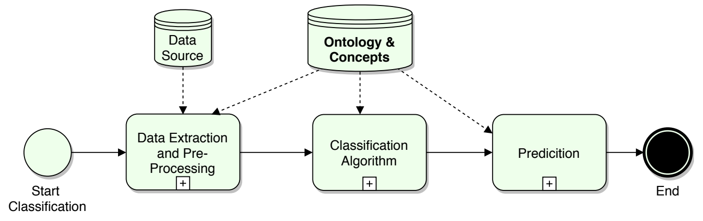


Figure 4: Unsupervised classification workflow

niques can be used. (Wijewickrema & Gamage, 2012a, 2012b)

The methods discussed further are not exclusive for ontology-based classification systems.

Multiclass classification

Ontology based classification are usually often a multiclass classification problem, where the thing to be classified is related to a concept described in the ontology. Strategies for solving multiclass classification problems includes:

- Naive Bayes[10][11]
- Decision Trees[11][12]
- Support Vector Machines [13]
- k-nearest neighbours [14]
- Neural networks [15]

Multi-label classification

In the case where a classification allows an item to be related to multiple concepts, we have a multi-label classification problem. A typical scenario for a multi-label classification problem is a compound entity such as a dataset which includes information that relates to difference concepts in a domain ontology. Strategies for solving multi-label classification problems are

- k-nearest neighbours [16]
- Decision Trees [17]
- Neural network [18]
- Boosting. From Zhou Zhi-Hua (2012)[19] "The term boosting refers to a familty of algorithms that are able to convert weak learners to strong" learners.

2.3 Semantic Similarity Measures

Two things that means the same or describe the same context is said to be semantically similar. In order to evaluate if two concepts or terms are semantically similar, different methods of measurements has been developed. In ontology based semantic search applications, semantic similarity measures are essential.

There are numerous methods that has been developed for measuring the similarity and relatedness of concepts. Similar concepts are ones that might be used interchangeably. For instance, a ’car’ and a ’bike’ shares similar properties as to be a device for transporting someone from A to B. Relatedness can be be used to measure the interconnectivity between concepts such as things or events. For instance, road conditions are related to the weather and accidents might be related to road conditions.

The ontologies/taxonomies representing the domain knowledge is organized as RDF graphs. Each concept of the domain is represented as a node in the graph. The links between nodes are edges. A common classification structure is to the most abstract concept on top, and then defined child-nodes to be specializations of that concept, with an is-a relation to the parent node. The deeper down the graph the concept is defined, the more concrete and specialized it is. Similarities can be measured when is-a relations are being used. Other kinds of relations, such as ’causes’ and ’has-a’ cannot be used for measuring similarity, however they can be used for measuring relatedness.

From literature, there are 3 dominant methods for measuring similarity: Purely path based, path and depth based, and path and information content (IC) based. Path based methods only consider the distance between nodes on the graph (by counting the number of edges needed to travel to get from one concept to the other). (Rada et.al., 1989) (Caviedes & Cimino, 2004) Path and depth based methods considers not only the distance between the concepts, but also how far down in the hierarchy it is defined. The intuition is that the more concrete and specialized two concepts are, the more similar they are, given that the distance is equal. Wu & Palmer, Leacock & Chodrow, Zhong et al, Nguyen & Al-Mubaid. The last categoriy is also path based, but includes contextual information. The main idea here is that each concept carries information. The more information shared by two concepts, the more similar they are. Take the example of a ’car’ and and ’bike’ again. If each concept carries information about what function the two concepts have, we see that they are similar. (Resnik, 1995, Jiang&Conrath, 1997, Lin, 1998)

The comparisons used here is presented in the review paper "Description and Evaluation of Semantic Similarty Measures Approches" [20]

Structure-based measures

Structure-based measures works by counting edges in a hierarchically structured ontology (a.k.a. taxonomy). The edges are *is-a* relationships between parent and child nodes.

Shortest Path

The most naive structure-based measure is simply computing the distance between two concept (the number of edges that separates them) [21]. Let C_1 and C_2 be two concepts. See Equation 2.1

$$Sim(C_1, C_2) = SP \tag{2.1}$$

A variant of 2.1 takes into account that there can be a number of paths between C_1 and C_2 and weights the similarity towards the longest path (max) between the two concepts.

$$Sim(C_1, C_2) = 2 * max(C_1, C_2) - SP \tag{2.2}$$

Wu & Palmer

A concept that resides deep down in the hierarchy tree of an ontology is more concrete that one that is defined higher up. It is therefore sensible to assume that two concepts defined deeper in the hierarchy are more closely related than two concepts defined higher up, if the node distance is the same for both pairs. Wu&Palmer takes this into consideration [22]. Wu&Palmer define the least common subsumer (LCS) as the most concrete concept that both C_1 and C_2 are specializations of. The $depth(C)$ function counts the path length from a concept C to the root of the hierarchy tree. From this the similarity function is defined as:

$$Sim_{W\&P}(C_1, C_2) = 2 \cdot \frac{depth(LCS)}{depth(C_1) + depth(C_2)} \tag{2.3}$$

The dep *depth* of the concepts, i.e. how deep down in the hierarchy tree [22]

Leacock Chodorow

Leacock & Chodorow developed a similarity measure for use with WordNet[23] which considers the shortest path $SP_{C_1C_2}$ between two concepts C_1 and C_2 scaled to the total depth D_{tot} of the hierarchy tree.

$$Sim_{LC}(C_1, C_2) = \log(\frac{2 \cdot D_{tot}}{SP_{C_1C_2}}) \tag{2.4}$$

Information Content based measures

Concepts which includes some information content (IC) can be compared with other concepts by measuring the similarity between the information contents. This method does not measure distances between concepts as the edge the based methods discussed in the previous section, but rather just look at the available information of each concept.

Feature-Based Measures

A concept described by a set of features (properties that makes the concept. As an example, a concept **car** may contain features that describes how a car can be useful (for personal transportation), or by its physical features (wheels, breaks, lights etc). Different concepts may share some of the same features. From this, similarity can be measured by comparing sets of features between concepts.

Hybrid Measures

Hybrid measures combines elements from the above similarity measurement methods, to make more robust and accurate measurements. Thabet Slimani[20] shows that hybrid-based measures offers the highest accuracy in a comparison of edge counting, information content, feature-based and hybrid semantic similarity methods on WordNet.

2.4 Semantic Interoperability

Allowing software systems to exchange information based on the context or meaning is called semantic interoperability. Different software systems agree on a model for representing context, which ensure unambiguous interpretation of the contents. The main difference from traditional syntactical information exchange is that the model is explicit, i.e. can be defined outside of the domain application, and shared between different applications.

Figure 5 illustrates the key difference between the traditional method of exchanging information and a hypothetical semantic framework. An simple example could be that A calculates a temperature profile that is to be read by B. In the syntactical example, A would simply encode the temperatures according to the specifications of a file format and write the data to file. B would need to know where to find the datafile, and how to correctly interpret the contents. In the case where the temperature unit is not stored as a part of the data, the two applications would have to agree on a standard for interpretation. For the semantic data exchange example, A would use a model that describes what a temperature profile is, in terms of the number of elements, data types, units etc. A would instantiate a datatype based on the model and submit it to the framework.

The SOFT framework [24] is a framework for semantic interoperability that works like the illustrating example in figure 5. SOFT is not based on web technologies such as OWL/RDF to build the information models, but uses a "simplified" approach that is more approachable for scientist with limited software science background. A translation of the SOFT information model into a web ontology is feasible, however due to the lack of standards and consensus on tools and models, the reward for doing this is limited. The thesis author is also the lead developer and designer of the SOFT framework.

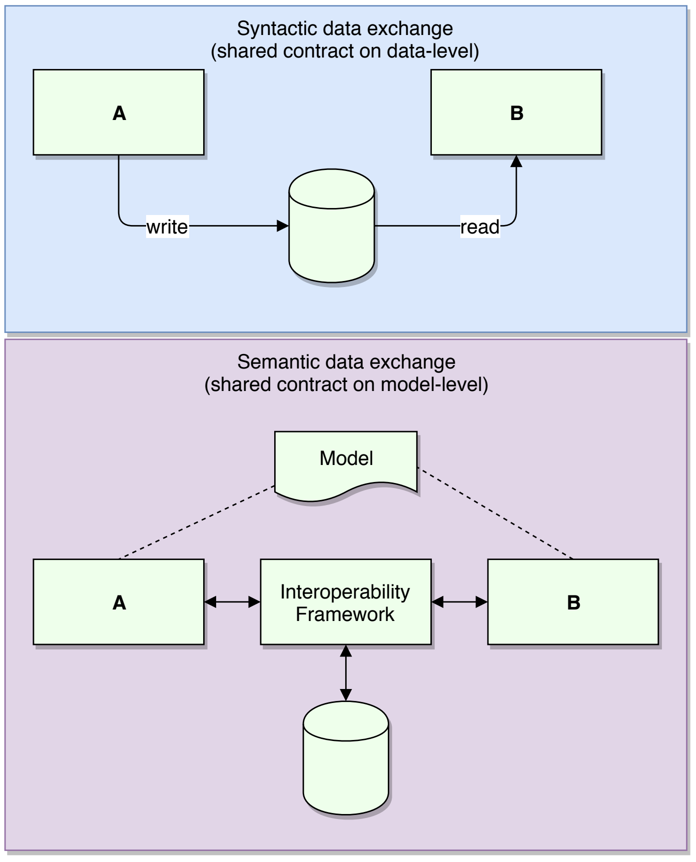

Syntactic data exchange
(shared contract on data-level)
A
B
write
read
Semantic data exchange
(shared contract on model-level)
Model
A
Interoperability Framework
B

Figure 5: Syntactic vs Semantic interoperability

3 Ontology Development *(RQ1)*

Introduction

The term *ontology* as used in computer science (and especially in the context of the semantic web) is simply a model of a domain (the domain is this case being transportation). This model contains elements (or things) that exists in that domain and the relationship between them. The language for representing ontologies is RDF (Resource Description Framework) (3). The distinction between and ontology, a vocabulary and a taxonomy can often be unclear. Here we use the term vocabularies on reusable models that defines concepts and relationships that are building blocks for other vocabularies or ontologies. The term taxonomy is used to express a specific relational structure that defines a classification tree.

Technologies

RDF

RDF[25] is a set of W3C recommendations that defines a standard for data interchange. RDF defines links between *resources*[1] in a structured called *triples*. A triple consist of a subject, a predicate (the relationship) and an object. From this basic structure, vocabularies and models can be defined. For instance, the friend-of-a-friend (foaf) vocabulary defines a class *Person* with properties such as *firstName* and *lastName*. It is then possible to use this model to describe a person.

[1]A *resource* can be a anything, like a concept, defined by unique URI

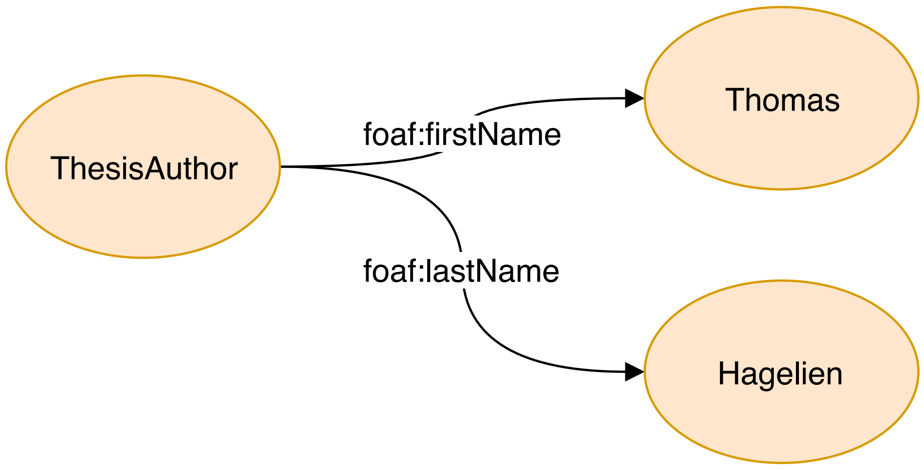


Figure 6: An example of a very simple RDF model describing the first and last name of the "thesis author"

There exists several textual representations for which RDF can be written. The RDF/XML syntax[26] is a W3C recommendation. By using RDF/XML the above model can be writtes as:

```
<foaf:Person rdf:about="#ThesisAuthor"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:firstName>Thomas</foaf:firstName>
  <foaf:lastName>Hagelien</foaf:lastName>
</foaf:Person>
```

3.0.1 SKOS

Classification schemes, thesauri and taxonomies are examples of knowledge organization systems (KOS). Best practises for representing a KOS has been developed by W3C since 2003. The latest standard (of August 2009) is called the Simple Knowledge Organization System (SKOS). SKOS is a common data model for sharing and linking knowledge organization systems via the Web. [27]. The greatest benefit of adopting this standard is that of interoperability between different applications and domains.

3.0.2 DCAT-AP

DCAT-AP vocabulary (DCAT application profil for data portals in Europe) (Figure 7 is an specification based on W3C's Data Catalogue Vocabulary (DCAT) for describing the metadata of public sector datasets.

WordNet

WordNet[30] is a database of the English vocabulary (lexical database). The following description is an excerpt from the WordNet webside[23].

> "WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (*synsets*), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. [...] WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing.
>
> WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity."

OrdVev

OrdVev is a Norwegian WordNet, developed by Kaldera Språkteknologi AS on behalf of the Norwegian national library. OrdVev is based on the danish DanNet, and has approximately 50.000 synsets (sets of synonyms). OrdVev can be dowloaded as a set of RDFs with documentation.

Development

The basic idea of employing an ontology in the semantic search methodology is to use the ontology as a knowledge graph of the domain, which links external datasets to a set of related concepts. An example of this is a description of a dataset containing bus timetables. This dataset may be linked to concepts such as "bus routes" or "travel information". It is possible to have multiple links from different concepts to a single dataset. Likewise from a single concept to multiple datasets. When a user issues a search query, the query is processed to identify the closest matching concepts. The matching concepts might not be linked with any datasets, however due to their position in the classification tree (knowledge graph), it is possible to identify similar concepts that do have dataset links.

In order to evaluate the effectiveness of ontology based search methods, an ontology had to be adopted or developed. As part of the Open Transport Data project at SINTEF Digital, an ontology with relevant concepts had been developed by Dr Marit K Natvig et.al. The highest level concepts are shown in figure 8.

To evaluate the suitability of the ontology for use as basis in semantic search, the following section gives a brief analysis

3.1 The original OTD Ontology

In the Open Transport Data Project (1.1) an example ontology has been developed. This ontology was initially evaluated for being used in the scope if this thesis. The following sections discuss the suitability of the OTD ontology. The approach described in following chapters is general, and can be applied to other ontologies as well. The approach is not limited to the transport domain.

A Brief analysis of the original OTD ontology

The OTD ontology have a set of concepts related to the transport domain. The relations between concepts are /is-a/ relations, with a single inheritance path. As seen in figure 9, the equivalence relationship are used to define synonyms. The meaning of each concepts are described in natural language using the rdfs:comment property. There is a total of 14 top-level concepts.

The OTD ontology identifies a large set of important concepts for describing different characteristics of the transport domain. The ontology is designed with a specific purpose in mind, and was not intended as a general extendable taxonomy. Therefore it lacks some characteristics to make it suitable for our purpose. What makes the ontology difficult to use as a taxonomy, is the lack of a stronger hierarchical structure where related concepts are classified under the same parent concepts. For instance, all concepts related to service, like "Transport Service", "Transport Service Characteristics" and "Transport Service Deviation" falls naturally under the same category. Having many specialized concepts as top-level concepts leads to an increased number of new top-level concepts when the taxonomy is extended. This is because there is no abstract concepts which serves as a classification group. The second drawback is the lack of formal annotation labels that puts the concept in the correct domain context. The classname of the concept itself, supported with an informal text comment, is the only hint of the purpose of the concept. This is especially problematic in conjunction with the previously discussed issue with multiple top-level concepts, which makes automatic machine navigation and identification difficult.

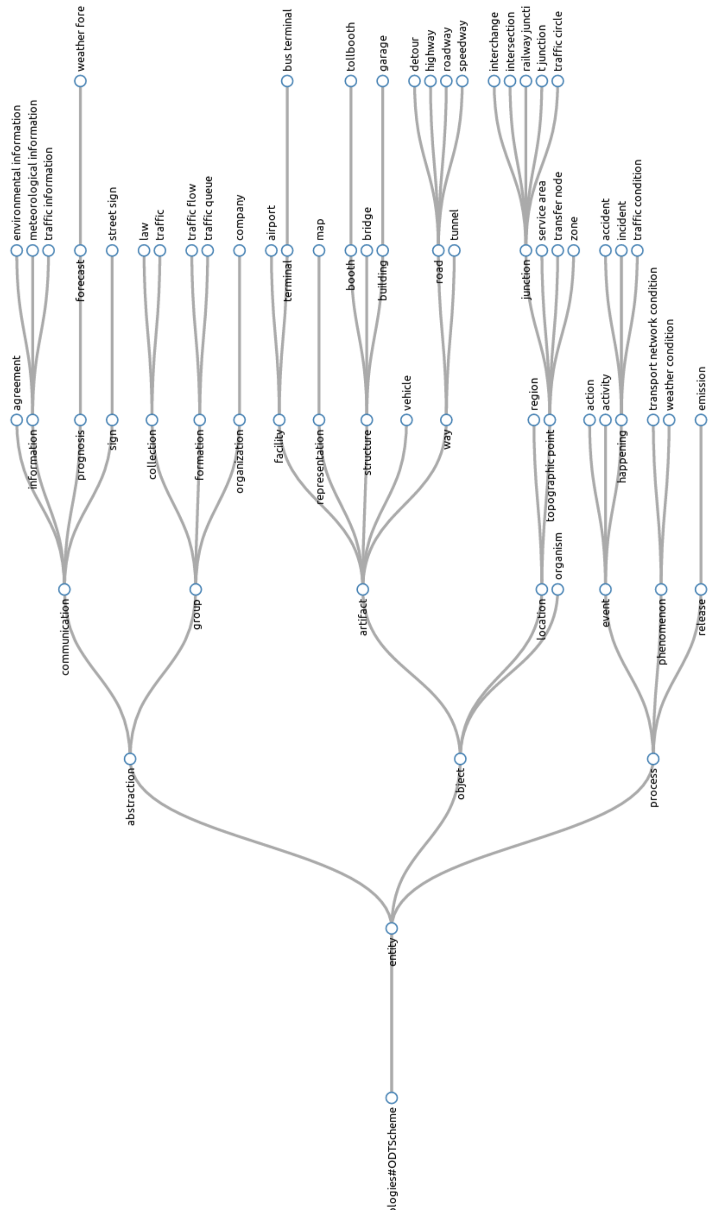


Figure 10: The Revised Open Transport Data Ontology (OTD2) is a proof-of-concept ontology containing a small subset of the possible concepts in the transport domain. The ontology is based on a small and abstract upper level ontology. This enables interoperability with other ontologies and simplifies adding extensions. All relations as between concepts are *is-a* relations, which makes the ontology suitable for classification.

Development of a Revised Open Transport Data Ontology (OTD2)

To overcome the limitations of the initial version of the ontology, a new ontology was developed. The main design criteria for the ontology was that it should be general enough for allowing concepts to be linked with any transport related datasets. This is directly addressing the first research question: "*How can the design of transport ontologies be improved to support the classification of current and future dataset*". To achieve this, the upper level concepts need to be general enough to capture a wide range of different context.

Another design goal for the new ontology was to make it compliant with a well-established standard. Many special purpose ontologies cannot easily be used in other contexts such as for interoperability with other domains. By basing the ontology on SKOS[27], it is compliant with a set of visualization tools and content validators.

Upper level ontology

Organization of knowledge and classification structures have large impact on the usefulness of ontologies and the success of its applications. Sadly, there are very little research on the theoretical foundations on ontology engineering. Best practises and principles have been presented by Arp, Smith and Spear in "Building Ontologies with Basic Formal Ontology (BFO)" [31], where they argue that an upper level (highly abstract) ontology strengthens the ability for supporting information retrieval, analysis and integration with other domains. In addition to BFO there exists a large set of other upper ontologies Business Objects Reference Ontology BORO[32], COSMO[33], Gist, SUMO, DOLCE and GFO to name a few. All of these are designed around the same fundamental principles; to maximize context coverage with few and unambiguous concepts. WordNet is itself based on an informal upper level ontology as it defines linguistic concepts up its most abstract/generic form.

The upper level ontology developed to support Open Transport Data is strictly based on *is-a* relations. Note that this upper level ontology is not designed to be generic for all fields of information organization. The existing categories have been purposely defined to fit a subgroup of known concepts.

The top level node of the ontology is *entity*. This is the most generic and abstract concept from which all other concepts are sub-types of. At level there are three unambiguous categories.

- **object** are physical things that exists.
- **process** defines the actions that makes changes over time.
- **abstraction** are non-physical things or ideas

Labels as content information

The revised ontology is based on SKOS, which has support for expressing content information through the use of *labels*. There are three types of labels defined in the SKOS vocabulary: the preferred lexical labels, the alternative lexical labels and the hidden lexical labels. The intended purpose of the different label types are somewhat weakly defined in the standard. For the purpose of the transport ontology, the labels are used to annotate each concept with terms that allows the concepts to be compared with lexical tools such as WordNet and OrdVev.

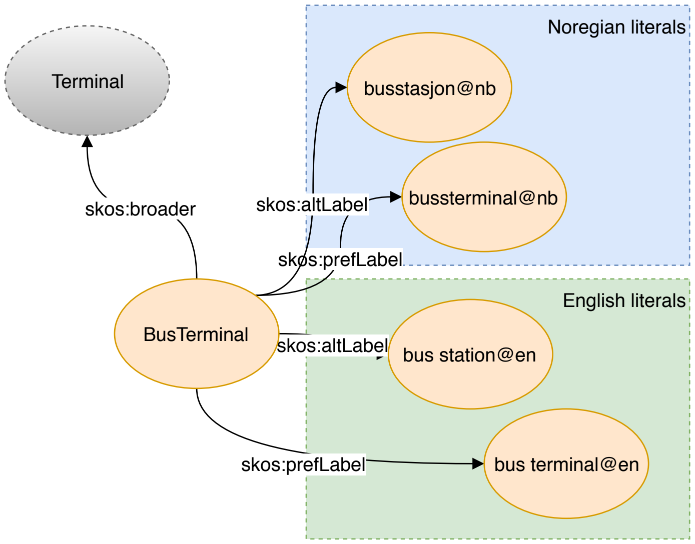


Figure 11: An example of using concept labels. Here the concept *BusTerminal* has labels in both norwegian and english. The labels can be compared with other terms using WordNet for the english labels, or OrdVev for norwegian labels. The labels can be compared with terms extracted from dataset descriptions, or with search queries written in natural language. Since the concept has labels of multiple languages, different languages can be used for both classification and during search.

Listing 3.1: The BusTerminal concept encoded as RDF/XML contains the formal information depicted in figure 11

```
<rdf:Description rdf:about="OTD#BusTerminal">
    <skos:altLabel xml:lang="nb">busstasjon</skos:altLabel>
    <skos:prefLabel xml:lang="nb">bussterminal</skos:prefLabel>
    <skos:inScheme rdf:resource="OTD#ODTScheme"/>
    <skos:prefLabel xml:lang="en">bus terminal</skos:prefLabel>
    <skos:broader rdf:resource="OTD#Terminal"/>
    <rdf:type rdf:resource="SKOS#Concept"/>
    <skos:altLabel xml:lang="en">bus station</skos:altLabel>
    <rdf:type rdf:resource="OWL#NamedIndividual"/>
</rdf:Description>
```

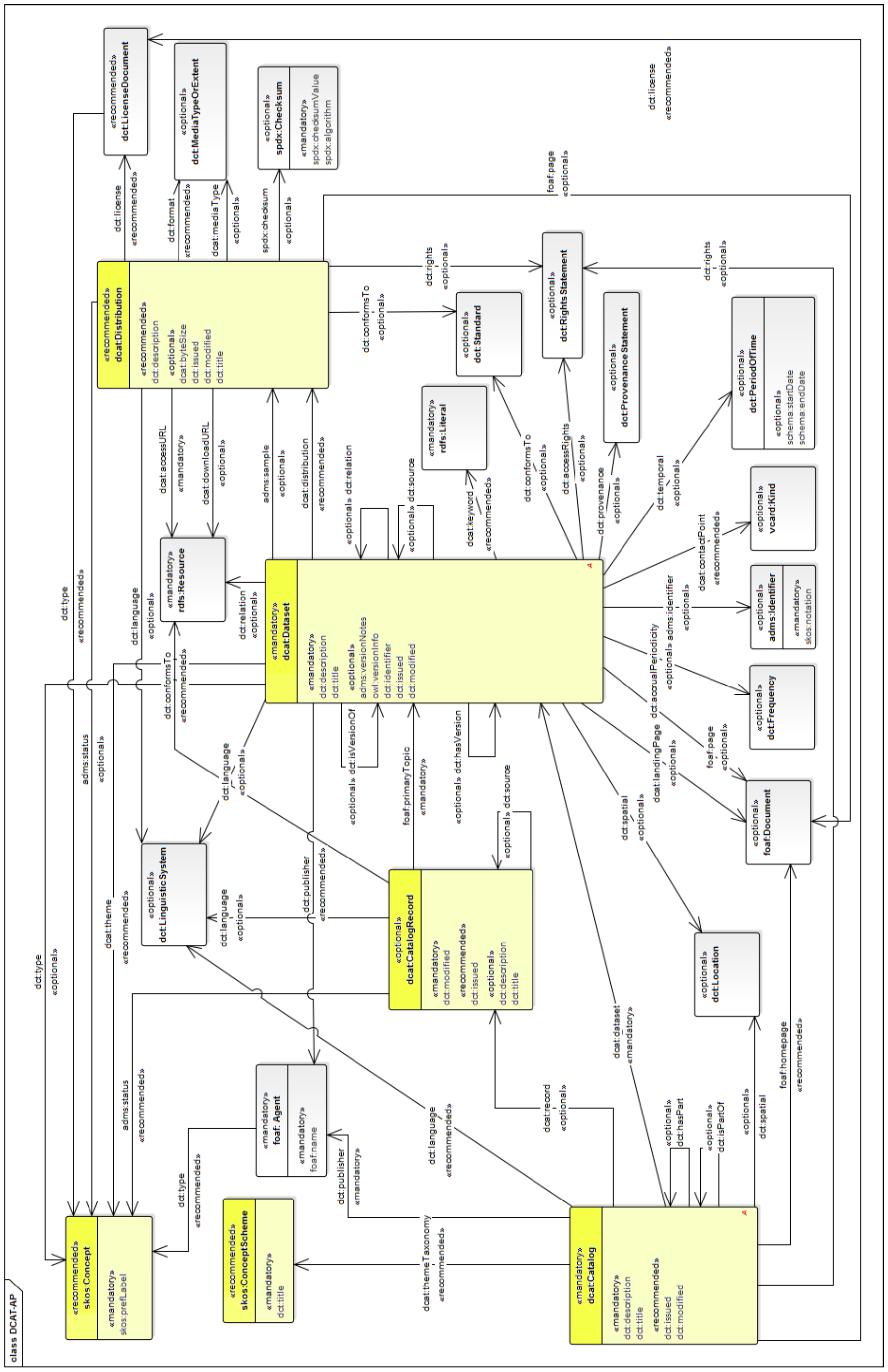

class DCAT-AP
«mandatory» dcat:Catalog
«mandatory» dct:description dct:title
«recommended» dct:issued dct:modified
«mandatory» dcat:Dataset
«mandatory» dct:description dct:title
«optional» adms:versionNotes owl:versionInfo dct:identifier dct:issued dct:modified
«recommended» dcat:Distribution
«recommended» dct:description
«optional» dcat:byteSize dct:issued dct:modified dct:title
«optional» dcat:CatalogRecord
«mandatory» dct:modified
«recommended» dct:issued
«optional» dct:description dct:title
«recommended» skos:Concept
«mandatory» skos:prefLabel
«recommended» skos:ConceptScheme
«mandatory» dct:title
«mandatory» foaf: Agent
«mandatory» foaf:name
«mandatory» rdfs:Resource
«optional» dct:LinguisticSystem
«recommended» dct:LicenseDocument
«optional» dct:MediaTypeOrExtent
«optional» spdx:Checksum
«mandatory» spdx:checksumValue spdx:algorithm
«optional» dct:Standard
«optional» dct:Rights Statement
«mandatory» rdfs:Literal
«optional» dct:Provenance Statement
«optional» dct:PeriodOfTime
«optional» schema:startDate schema:endDate
«optional» vcard:Kind
«optional» adms:Identifier
«mandatory» skos:notation
«optional» dct:Frequency
«optional» foaf:Document
«optional» dct:Location

Figure 7: The DCAT Application profile for data portals[28] is a European standard for describing (public sector) datasets. The specification is based on the Data Catalogue Vocabulary (DCAT)[29]

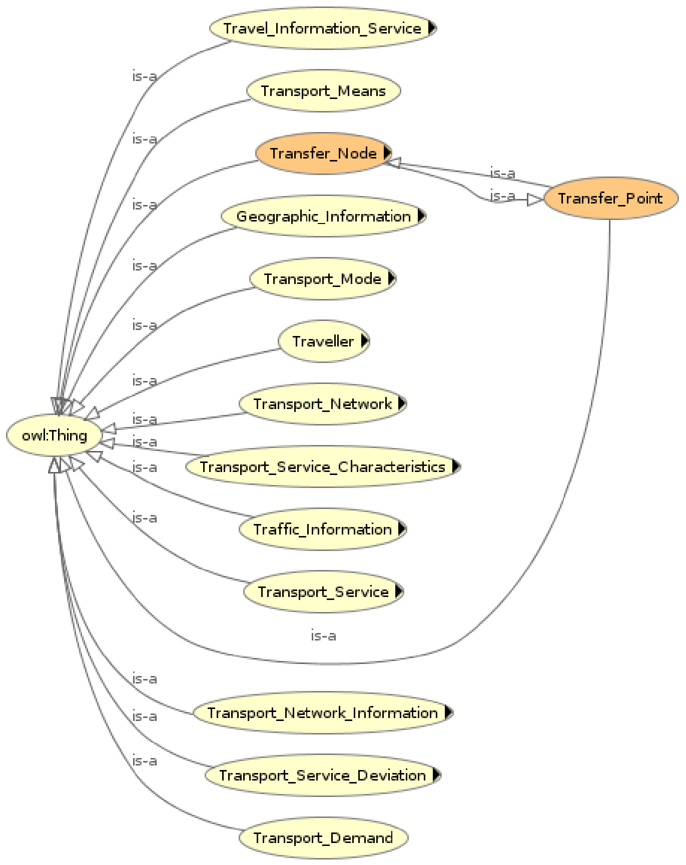


Figure 8: The Original Open Traffic Data Ontology defines specific transport related concepts. It is, however, not suitable as general ontology for representing any transport related dataset.

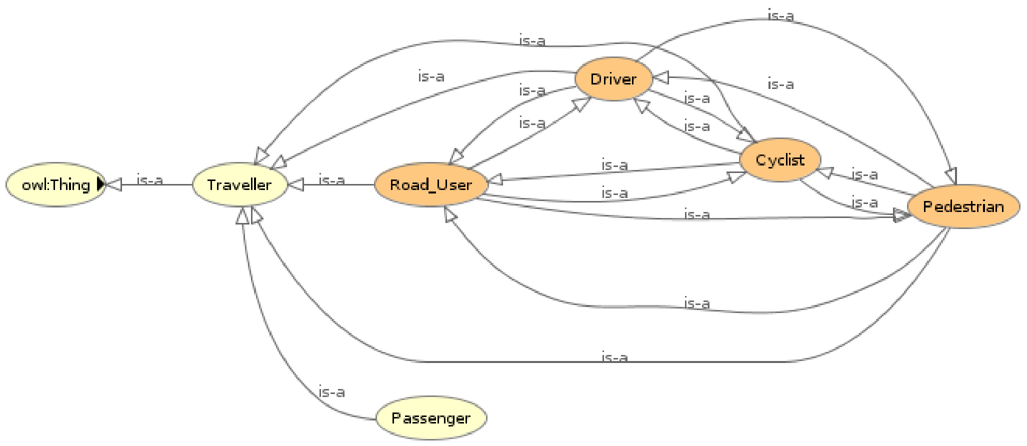


Figure 9: The OTD Ontology subdomain of a Traveller. Equivalent or similar concepts are modelled using *it-a* relations

4 Tagging and Classification*(RQ2)*

In this chapter, a hybrid classification method is developed to examine the research questions posed in the section 1.3: "Can an automatic- or semi-automatic classification process be developed to replace manual tagging?" and if semantic technology can be applied to improve the discoverability of open transportation datasets.

4.1 Technologies

Term Frequency-Inverse Document Frequency (TF-IDF)

In computational linguistics, a common heuristic method for classifying documents is to identify words (terms), used in the documents, that have a strong semantic relation to the document context. Words with a strong contextual dependency tends to be appear more frequent in relevant documents, and more seldom or not at all in documents with little or no relevance. Identifying the most significant words is a two-step process. The first step is to count the frequency of every word in all the documents. Term Frequency ($TF_{w,D}$) is the number of times a word w occurs in a document D. Common words such as 'and', or', 'from' etc might have a high frequency, but since they occur equally frequent in all documents, they- have little contextual relevance. The second step of the process is to count the number of documents a word (w) occurs in. This is called the document frequency (DF_w). By dividing the total number of documents by the document frequency, we get a number from 1 (the term occurs in every document) to the total number of documents (if the term occurs in only one document). Taking the logarithm of this number gives a reasonable weight factor that discriminates words that are used across a large fraction of the total set of documents from those that are not. This factor is called the inverse document frequency (IDF). There are different weighting schemes based on TF-IDF. The simplest is to multiply the term frequency with the inverse document frequency.

$$W_{w,D} = TF_{w,D} \cdot \log \frac{|D|}{DF_w} \tag{4.1}$$

OTD Similarity

In order to allow for a dataset to be linked to a concept with a value indicating how similar the two resources are, the a simple datamodel called *OTD Similarity* has been developed.

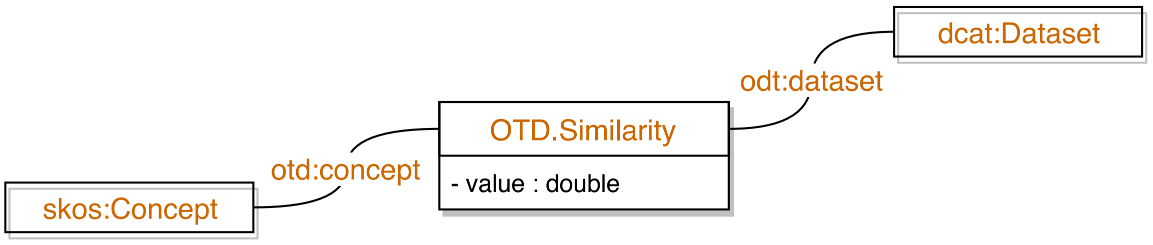


Figure 12: A simple model for linking and expressing the semantic score between a SKOS concept and a DCAT dataset

Concept Vectors

Concept vectors are vector representation of concept properties. The vector length corresponds to the number of concepts it represents. The first element represents a value related to the first concept. The second element by a value related to the second concept, etc. In figure 13, the elements of the concept vector corresponds to the semantic similarity score between a dataset and a concept node defined in an ontology. The similarity score between dataset D_1 and concept C_1 is 0.2. The first element of the vector is thus 0.2. This continues for every element of the vector.

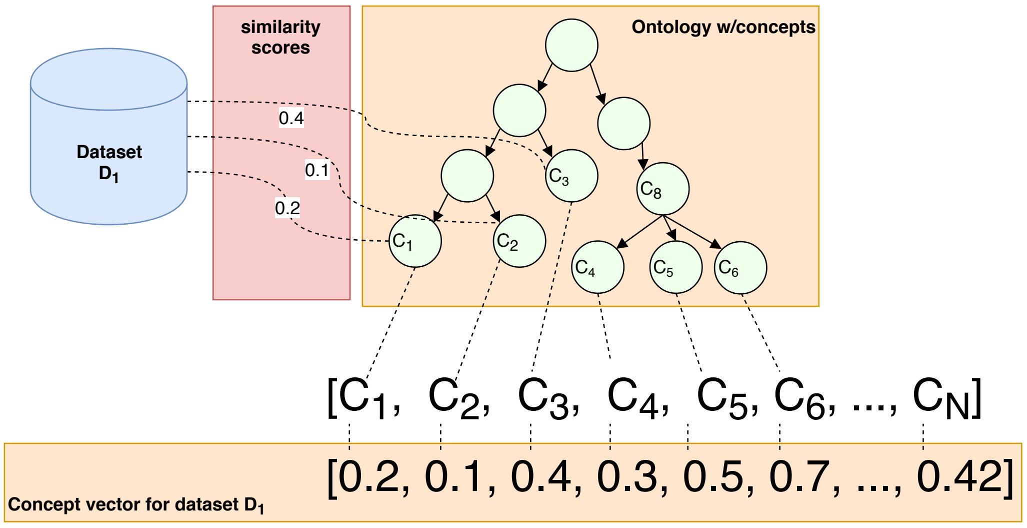


Figure 13: A concept vector is constructed from the similarity scores (i.e semantic relation between (in this case) a dataset and a concept)

4.2 Classification Method Evaluation

The strategy towards defining a method for dataset classification had the following constraints:

- A CKAN server has been set up by SINTEF digital with a set of harvested dataset descriptions. The main language for describing the datasets were written in Norwegian.
- The available datasets relevant for the transportation domain was limited in number.
- The only directly available information about the dataset is via the DCAT dataset descriptions.

Preparing text from dataset descriptions

The text from dataset descriptions needs to be preprocessed before it can be used for classification. An excerpt from a DCAT data description is shown in listing 4.1

```
<rdf:RDF>
    <dcat:Dataset rdf:about="http://78.91.98.234/dataset/caefad21-41cc-
        <dcat:keyword>elbil</dcat:keyword>
        <dcat:keyword>ladestasjon</dcat:keyword>
        <dct:description>Datasettet viser plasseringen
            til ladestasjonene for elbiler i Stavanger
        </dct:description>
        <dct:identifier>caefad21-41cc-41e6-9f59-e050486686ea
            </dct:identifier>
        <dct:title>
            Ladestasjoner for elbiler i Stavanger
        </dct:title>
    ...
```

Listing 4.1: An except from a dataset description retrieved from the SINTEF DCAT server

The title and description fields are used in further processing. Quite often the natural language descriptions found in dataset annotations will contain markup (for instance HTML or XML). This needs to be removed. Further, the sentences can be divided into individual words (or tokens). Words such as: 'a', 'is', 'at', 'the', etc are called stop words. Stop words do not contribute with any contextual meaning, and should be removed. It is also possible to go even further and correct or ignore potentially misspelled words, and normalizing words such as transforming plurals into singulars etc.

Attempting TF-IDF + SVM classification

Standard methods for document categorization was initially attempted. A text document was extracted from the title and description fields of the DCAT metadata.

The entire process from data extraction to classification is illustrated in figure 14

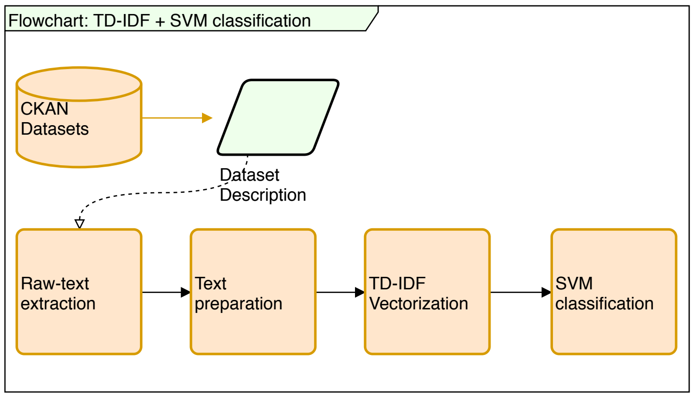


Figure 14: A flowchart describing the process of text classification from retrieving the data from a database, through preprocessing and vectorization to the classification step.

The term frequency-inverse document frequency (TF-IDF) was employed to extract features from the text. A support vector machine (SVM)[34] classifier was then used to predict relevant concepts. Preliminary results shows that important features relevant for the transportation domain was not considered important(got weighted low) by the TD-IDF vectorization. This was mainly due to a combination of two factors. The written descriptions did not emphasize the relevant features. In addition to very general description, some were even repeated throughout several datasets.

With this, the conclusion was that the method of using state of the art text classification methods were not suitable due to the limitations mentioned above.

Lexical similarity

The concepts defined in the ontology (see section **??**ontology-development) is labeled with words and phrases that can be directly compared with words and phrases in a description text. Lexical similarity measures how similar two sets of words are. From this, a naive assumption can be made: if the phrases used to describe a dataset is similar to labels that describes concepts in the ontology, the dataset is related to the concept.

4.2.1 Automatic information extraction from heterogenous datasources

SKOS Vocabulary

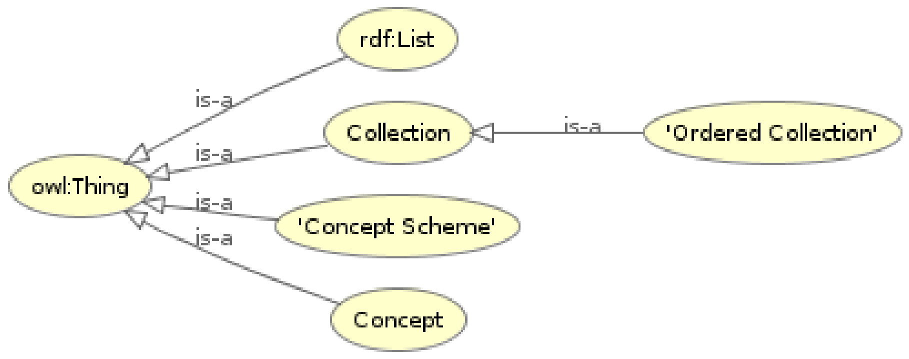


Figure 15: SKOS Classes

The SKOS (fig 15) vocabulary defines semantic relations between concepts using the categories hierarchical and associative. Hierarchical relations defines categories in terms of ancestors (broader), or child (narrower), much like a taxonomy would be constructed using is-a relationships. Associative relations indicates if two concepts are related. However, the SKOS does not support way to quantify the degree of associative relationships. To link a concept in the ODT ontology to a dataset, there is a need to say something about the importance of a relationship as a score from 0 (not related) to 1 (most related). For this purpose a new vocabulary (ODT-Similarity) is developed as part of this thesis. The ODT-Similarity builds on the DCAT and SKOS vocabulary. As shown in figure **??** the odt:Similarity class is linked to a skos:Concept and a dcat:Dataset. Note that in DCAT-AP, the dcat:Dataset dervices from a skos:Concept. This is fine, as long as we do not confuse the dataset-definition with the SKOS concepts defined in the ODT ontology. A link between a skos:Concept and a dcat:Dataset is given a a value/score (odt:score). This score indicates how related a dataset and a concept is. If the score is 0.0, there is no link between the two. If, however, the score is 1.0 (100%) there is a 1:1 relation between the concept and the similarity. Note that multiple instances of odt:Similarity can be used to link a single dataset to the ontology.

Classification

Automatic classification requires that correct and formal information about the data contents exists, which is often not the case. There are not guarantees about the file formats either. This leaves us with manual tagging as the most accurate method of classification. This is, however, not always feasible given that manual tagging is a time consuming task. A hybrid classification scheme was developed as part of the thesis work. The hybrid classification scheme consist of the manual tagger, and four fully automated methods that exploits existing metadata:

- DCAT:keyword (comparing keywords with concepts)
- DCT:description (analysis of free-text record descriptions)
- DCT:format + DCT:distribtion (Using knowledge about known formats to infer concept relations)

4.2.2 CCS Matrix

An important property for our classification and search method is the ability to compare two concepts. An assumption is that datasets that are linked to concepts that are similar to each other are more likely to have commonalities than those that are linked to unrelated concepts.

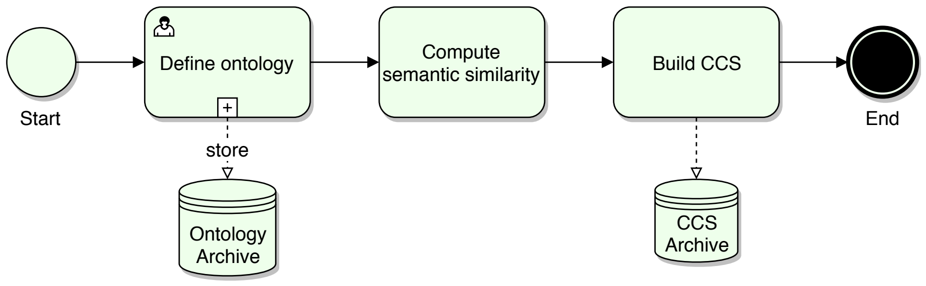


Figure 16: BPMN diagram - calculate CCS

To quantify the degree of similarity between two concepts c_1 and c_2, a similarity function $SIM(c_1, c_2)$ is constructed. More generally we have that the similarity between concepts c_i and c_j is $s_{ij} = SIM(c_i, c_j)$. Since the ontology changes rarely, the semantic similarity of all concepts can be computed and stored/cached as a matrix. The concept-concept similarity matrix (CCS) is a symmetric m-by-m matrix, where each element s_{ij} represents the semantic similarity between the concepts c_i and c_j.

Table 1 shows an excerpt of the transportation CCS matrix.

Table 1: Excerpt of the OTD Concept-Concept Similarity Matrix. Here, Wu & Palmer is used to compute the similarity scores

	otd.StreetSign	*otd.Information*	*otd.Organism*	*otd.TrafficCondition*
otd.StreetSign	1.000	0.667	0.250	0.200
otd.Information	0.667	1.000	0.286	0.222
otd.Organism	0.250	0.286	1.000	0.250
otd.TrafficCondition	0.200	0.222	0.250	1.000
otd.Bicycle	0.182	0.200	0.444	0.182
otd.Happening	0.222	0.250	0.286	0.889
otd.RailwayJunction	0.182	0.200	0.444	0.182
otd.TopographicPoint	0.222	0.250	0.571	0.222
otd.TJunction	0.182	0.200	0.444	0.182
otd.Way	0.222	0.250	0.571	0.222
otd.Forecast	0.600	0.667	0.250	0.200
otd.Event	0.250	0.286	0.333	0.750
otd.Organization	0.444	0.500	0.286	0.222
otd.TrafficFlow	0.400	0.444	0.250	0.200
otd.Speedway	0.182	0.200	0.444	0.182
otd.Timetable	0.545	0.800	0.222	0.182
otd.TransferNode	0.200	0.222	0.500	0.200
otd.RoutePlan	0.182	0.200	0.444	0.18

In the research framework developed here, the CCS is computed and stored in a cloud hosted MongoDB service. The storage instance is identified with a unique identifier UUID[1] for future retrieval.

When a concept c_i is linked to a dataset d, weighted with a similarity score $s_{c_i d}$, the CCS can be used to compute the dataset relatedness for all concepts in the ontology. For instance, to compute the relatedness between the concept c_j given that the relatedness between dataset d and concept c_i is known, we have that:

$$s_{dc_j} = CCS_{ij} * s_{dc_i} \tag{4.2}$$

Listing 4.2: Python source code for computing the CCS Matrix based on a list of concepts and an ontology navigator which implements the similarity function sim_wup (Wu & Palmer similarity)

```
def compute_ccs(concepts, navigator):
    """
```

[1]A Universally Unique Identifier (UUID) is a long number (128-bit) used for the assigning a unique identification to some entity

```
    Description:
      The CCS matrix is computed by comparing
      all ontology concepts with each other
      by using Wu&Palmer similarity.

    Input:
      concepts - a list of concept types
      navigator - the ontology navigator

    Output:
      a pandas dataframe containing
      the CCS matrix
    """
    data = []
    for c1 in concepts:
        v = []
        for c2 in concepts:
            score = navigator.sim_wup(c1, c2)
            v.append(score)
        data.append(v)
    ccs = pd.DataFrame(columns=concepts,
                       index=concepts,
                       data=data)
    return ccs
```

4.2.3 CDS Matrix

In the previous section, the CCS was created as a lookup-table for fetching pre-computed semantic similarity measures between two concepts. This section introduces concept vectors CV, and discuss how they can be used to compare similarities between datasets. The concept dataset similarity matrix CDS is constructed by a set of concept vectors, and is an essential artifact for the semantic search ranking.

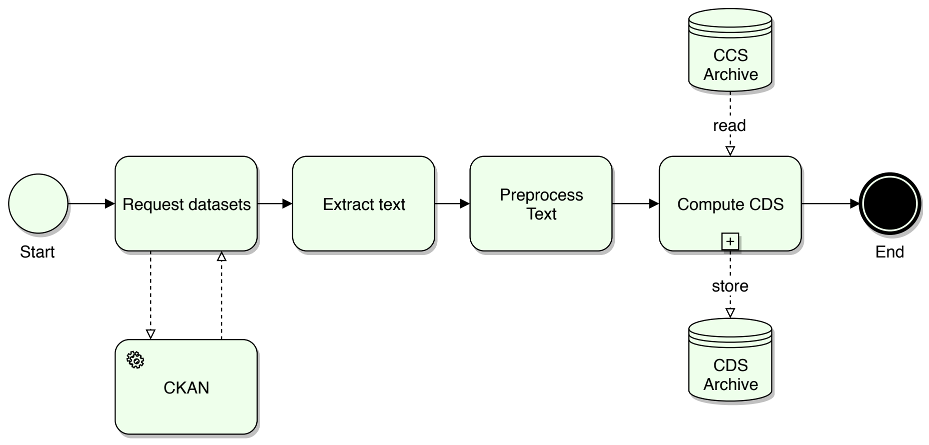


Figure 17: BPMN diagram - calculate CDS

Dataset descriptions are linked to concepts in the onotology, either manually or automatically using the Hybrid Classification Service. The outcome of this process is an *OTD.Similarity* graph (defined is chapter 4.1) that identifies a concept, a dataset description and a semantic similarity score (s-score) that annotates the relatedness/semantic similarity between the concept and the dataset. There can exist numerous links between concepts and datasets. There can be links from different concepts going to the same dataset, or links from a single concept to multiple datasets. Even contradictory s-scores may appear. This is because different classification strategies may produce different s-scores. If this is the case, the highest s-score is given precedence.

For each dataset that is linked to the ontology, a vector containing the s-scores for each ontological concept is constructed. This vector is unique for each dataset, and will be used for comparing datasets. Here we call this vector the Concept Vector (CV), although we could also have used the term s-score vector. The important thing is that each element in the vector contains a score representing the semantic similarity between the dataset and one specific concept.

Listing 4.3: Python source code for computing the Concept-Dataset-Similarity matrix

```
def compute_cds(concepts,
                datasets,
                ccs,
                similarity_graph):
    """
    Description:
```

```
    Compute the Concept-Dataset Similarity matrix.

  Input:
    concepts - A list of all concepts in the ontology
    datasets - A RDF graph of DCAT-AP datasets
    ccs      - The concept-concept similarity matrix
               similarity_graph - A RDF graph containing
               similarity scores between ontology
               concepts and datasets
  Output:
    cds      - A pandas dataframe containing the concept-dataset
               similarity matrix
  """
  cds = pd.DataFrame(None, columns=concepts)

  # Create zero-vectors for each dataset
      # The datasets are found by extracting the subjects
      # of the RDF graph which are of type DCAT.Dataset
      for dataset in datasets_graph.subjects(RDF.type, DCAT.Dataset):
          cds.loc[dataset] = np.nan

  # For each similarity entry, compute the corresponding value
      # for all the other ontology concepts based on the known CCS
      # score. This allows for closely related concepts to also be
      # relevant for the given dataset.
  #
  # If a concept has been assigned a higher score
      # already, the old score is kept.
      for similarity in similarity_graph.subjects(
            RDF.type,
            OTD.Similarity):
          dataset = next(similarity_graph.objects(
            similarity,
            OTD.dataset),None)
          concept = next(similarity_graph.objects(
            similarity,
            OTD.concept), None)
          score = (float)(next(similarity_graph.objects(
            similarity, OTD.score), np.nan))
          for cs in concepts:
              simscore = (float)(ccs[cs][concept]) * score
              cds.loc[dataset][cs] = np.nanmax(
                [simscore, cds.loc[dataset][cs]])
      return cds.dropna(thresh=1)
```

Let sim_{dc} be the s-score between dataset d and concept c. The concept vector d is: $CV_d = [sim_{dc_1}, sim_{dc_2}, sim_{dc3}, ..., sim_{dc_n}]$, where n is the total number of concepts defined in the ontology.

For the unknown values of sim_{dc} the equation 4.2 from the previous section can be applied to complete the vector.

The similarity between two dataset can be computed using the properties of the concept vectors. A common technique in data mining is to compute the cosine angle between the two vectors.

$similarity = cos(\theta) = \frac{CV_1 * CV_2}{\|CV_1\|\|CV_2\|}$

Example 4.2.1 *Comparing two datasets using the cosine similarity function.*

This example illustrates how the cosine similarity function can be used to compare two datasets.

The datasets have been classified and a semantic similarity factor has been computed for all concepts. In this example the ontology have only 5 concepts C_1, C_2, ... C_5.

The concept vectors for two datasets D_1 and D_2 is defined as $CV_{D_1} = [1, 0.2, 0.2, 0.8, 0.2]$ and $CV_{D_2} = [0.7, 1.0, 0.7, 0.6, 0.3]$. From this it can be observed that the first dataset is exactly match the first concept C_1 as the value is 1.0. It is also highly related to concept C_4 with a value of 0.8. The second dataset is exactly matching the second concept C_2.

ref to 4.2.1

4.2.4 Dataset Classification

Automatic tagging of datasets

One of the main goals of this project was to analyze methods for performing automatic classification of dataset. Different methods were examined, and in this section the findings are presented

Latent Semantic Analysis of the "Statens Vegvesen" dataset

In order to study the relationship between a set of dataset descriptions and the terms being used, latent semantic analysis (LSA) was employed. LSA is common technique in natural language processing [35]. LSA assumes that there exists a connection between words that share the same meaning and how frequently they occur in similar texts.

Text was extracted from the description fields in the Statens Vegvesen metadata. The text was cleaned up where XML tags was found. The next step was to build a vector consisting of a weighted factor for each term used in the text. Here the TF-IDF vectorization technique was employed, which ensures that terms without semantic significance is filtered out. In addition a set of stop words are also removed from the term set.

Singular Value Decomposition (SVD) is a technique for dimensional reduction to perform classification. The output from the SVD solver is a set of term clusters. If successful, each cluster would constitute an identifiable concept.

Some results from the analysis

Concept0	Concept1	Concept2	Concept3
nbsb	informasjon	eu	attraksjonene
www	tilbyr	eu kontroll	attraksjonene langs
vegvesen	tmc	godkjente	attraksjonene langs nor
trafikkmeldinger	norsk	kontroll	bilder
veg	informasjon parkeringsområder	kontrollorgan	bilder nasjonale
175	informasjon parkeringsområder tilbyr	oversikt	bilder nasjonale turistv
radio	parkering	ådt	engelsk tysk
nbsp www	parkering hele	trafikkmengde	inkluidert
175 nbsp	parkering hele landet	beregningen	inkludert attraksjonene
tjenesten	parkeringsiområder	hvert år	inkludert attraksjonene

Table 2: A small except showing the grouping terms discovered by running the singular value decomposition dimensional reduction on word vectors

Analysis

A few of the identified concepts makes sense for human interpretation. We can see that concept 0 relates to traffic information with words like radio, web, trafikkmeldinger. Concept 1 relates to data and news related to statens vegvesen. Concept 2 and 3 is about parking, and information about parking areas. Concept 4 is related to mandatory periodic vehicle inspections (EU-control). Concept 5 relates to traffic density. Concept 6 relates to road attractions/sightseeings. Concept 7 relates to vehicle types and fuel emissions. Concept 8 relates to routing plans and services. From here the concepts become more diffuse. However, it is interesting that without any guiding we have managed to identify the following concepts just from the informal text:

- News and Traffic reports

- Parking (ares, information, availability)
- EU-control (Mandatory vehicle inspections
- Traffic Density
- Main sights and attractions
- Vehicle types and fuel emissions
- Route plans and services

Automatic classification of datasets using latent semantic analysis looks feasible, however many of the classifications groups were quite bad.

Direct comparison of similarity scores

An intuitive approach to comparing a text with a set of concepts is to measure the semantic similarity between the terms in the text with the labels used to annotate each concept. Here the concepts from the ontology was used directly as classification groups. The text needed to be cleaned for noise and stripped for stop-words. The approach was to iterate through every word in the entire word-vector for the document/description field,and compare this to all the labels assigned for each concept. A decent model for comparing the semantic similarity between words is implemented in the WordNet taxonomies. Here, Wu & Palmer is used to compute word similarity by measuring the distance between two synsets in the WordNet taxonomy and adjust this based on the taxonomic depth of the least common subsumer (LCS). This will give a score between 0 and 1.

For the comparing datasets using the norwegian language, the OrdVev taxonomy is used. Here the ordvev taxonomy was downloaded from Språkbanken. The Wu & Palmer algorithm was implemented and applied to the OrdVev taxonomy.

4.3 Semantic Search

In this section, the search algorithm is described. Figure 18 illustrates the steps involved in the search process. The search process starts with the user typing a query string into the search field. The query string is processed using NLP to extract terms. Labeled concepts from the ontology is compared with the extracted terms using WordNet (or eqivalent lexical database for a different language). The output from the comparison process is a concept vector for the query. By comparing the query concept vector with the predefined concept/dataset similarity vector, a similarity score for each registered dataset is generated. After sorting and filtering, a ranking of the results can be presented.

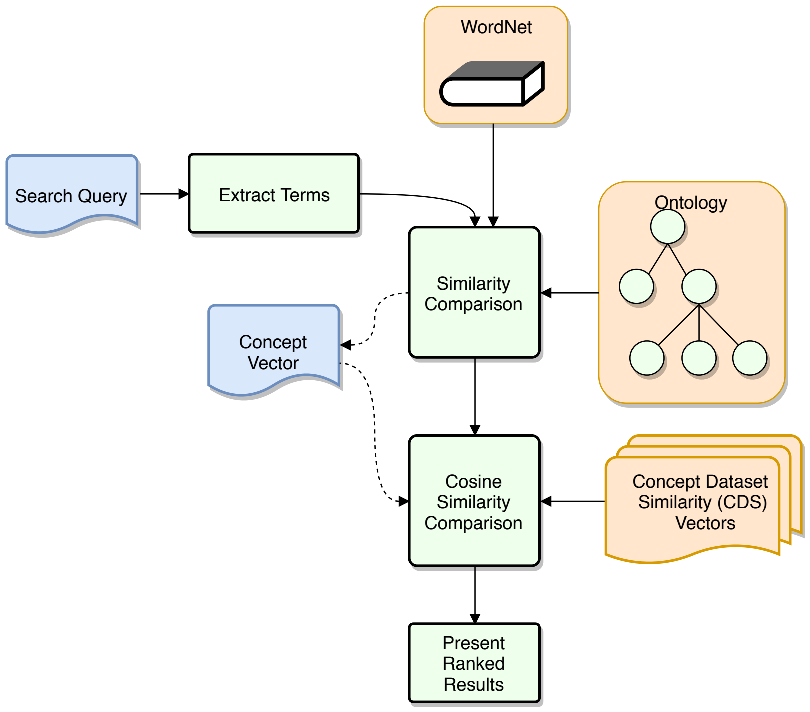


Figure 18: Flowchart of the search process. The figure illustrates the process of extracting terms from a search query. Concepts from the ontology is compared with the extracted terms using the WordNet dictionary. A concept vector that represents the search term is computed. The concept vector is then compared to the concept vectors of all registered datasets, by using cosine similarity. The results are finally presented in ranked order to the end user.

-

5 Semantic Interoperability *(RQ3)*

The exchange of information between software systems based on datamodels, which provide a shared and unambiguous meaning of the data, is called semantic interoperability. Instead of agreeing on a set of file formats or protocols to exchange data, software systems can share information by using a semantic framework as a mediator. As this topic is relatively new and often undiscovered field, there are currently no shared standards for such frameworks. Previously - the thesis author, Thomas Hagelien, has been the architect and main developer of the SOFT Framework (The SINTEF Open Framework and Tools), which is both a framework for semantic interoperability and a framework for model driven development used in a number of commercial and open software projects at SINTEF[24].

The key concept of SOFT is modelling the domain of an application using entities, and relationships between entities. The data models are then used by code generators to provide source code (for instance as classes that represents the entities), documentation and other artifacts. Through a set of application programming interfaces API's, it is possible to serialize data in and out from instances of the generated classes. The domain application will not need to know anything about storage formats or protocols. It only has to adhere to the APIs of the interoperability framework.

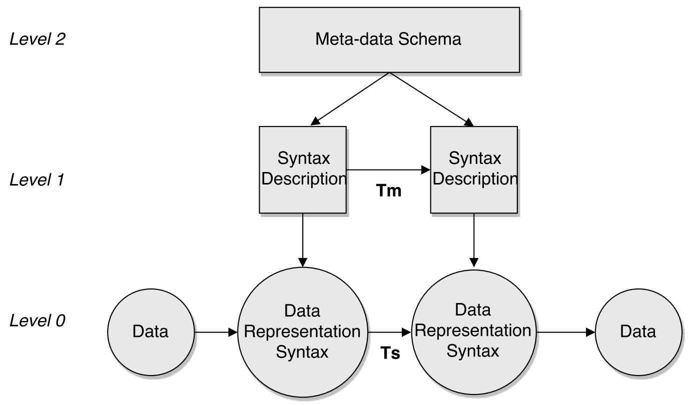
This method has several practical implications

- The domain code is more compact, as all data i/o, version handling and state representation is generated or provided by the framework
  - Easier to maintain
  - Easier to test
  - Quicker to develop
- Different codes can share information without worrying about the N-squared problem (which is the need to implemement translators between every software system)
- Existing data can be made available to numerous software systems by implementing a single driver and provide the relevant metadata.

5.1 Related work

The Eclipse Modeling Framework (EMF)[36] allows for the developer to create a data model and generate code or other artifacts based on the model. EMF is a mature and widely adopted framework. Different projects can share also data models. While EMF provides a very powerful programming paradigm for software developers, working with EMF and the modelling language requires deep knowledge in software modelling. A working scientist with limited programming skills is unlikely to pick up EMF to simplify her/his development workflow.

5.2 Model driven development



Data are raw facts which cannot be interpreted in any meaningful way unless information is provided to give context. Information is represented as formalized data with explicit or implicit relations to other pieces of data. Data used to construct information is called metadata. Formalization of metadata (formal metadata schema) is meta-metadata, and one can build layers upon layers of abstractions this way. For our purpose we define 3 layers, as indicated in figure 5.2. At the highest level we have a formal metadata schema which tells us how metadata can be formally expressed. This is needed for building model-to-model transformations (Tm) between level 1 metadata. At level 0 we have the actual data which has a set of values that can be structured syntactically. The transformation step (Ts) indicates that data given in the first syntactic representation is transformed to a different syntactic representation and the values (raw data) might have changed. All processes where a set of input data is read, and a new set of output data is produced can be represented with this figure. Instead of describing the transformation at a syntactic level, we can also describe this transformation at a meta-level. Given enough information about a domain (ontology) the transformation Tm can be inferred. In cases where the transformation is complex, this may not practical, but for transformation which involves simple filtering/interpolation steps or unit-conversion operations, this is feasible.

For automatic model transformations, the information that represents the relevant state of a domain application must be semantically equivalent or less specific than the available data. If the domain application requires more specific data than is available, or the data is simply semantically disjoint, the data cannot be used.

In the following sections the interoperability platform SOFT is presented. SOFT works at the domain application level and provides a metadata framework for representing software state. This knowledge will be merged with the domain ontology which describes the context of the open datasets.

5.2.1 SOFT

SOFT is an acronym for SINTEF Open Framework and Tools. SOFT5 is a set of libraries and tools to support scientific software development. The development of SOFT5 was motivated by many years of experience with developing scientific software, where it was observed that a lot of efforts went into developing parts that had little to do with the domain. A significant part of the development process was spent on different software engineering tasks, such as code design, the handling of I/O, correct memory handling of the program state and writing import and export filters in order to use data from different sources. In addition comes the code maintenance with support of legacy formats and the introduction of new features and changes to internal data state in the scientific software. With SOFT5 it is possible to utilize reusable software components that handle all this, or develop new reusable software components that can be used by others in the same framework.

The main components of SOFT5 is shown in Figure 5.2.1. The key modules are the tools, storage support and plugin framework. The key tools are the scripting utility and the code generator.

SOFT5 contains a core library with plugin support. The library also comes with set of interfaces (API) to create extensions and custom plugins. The core library is used to connect a software application with the framework.

There are currently two supported storage options for storing with SOFT5, namely HDF5 and MongoDB. Local data stored in HDF5 files is suitable for managing local data

The main approach to developing software with SOFT5 is to incrementally describe the domain of the software using entities (see below). The entities can represent different elements of the software, and be used in handling I/O as well as in code generation and documentation. Entities can also be used for annotating data and data sets. This might be useful in cases where for instance the origin of the data, license and ownership are of importance.

Since any complex software will have many entities and often multiple instances of the same entity, SOFT5 allows for creating collections of entities with defined relationships. These entity collections are called 'collections' (see below).

One idea of SOFT5 is that software may be written is such way that business logic is handled by the codebase, while I/O, file-formats, version handling, data import/export and interoperability can be handled by reusable components in the SOFT5-framework, thus reducing risk and development time.

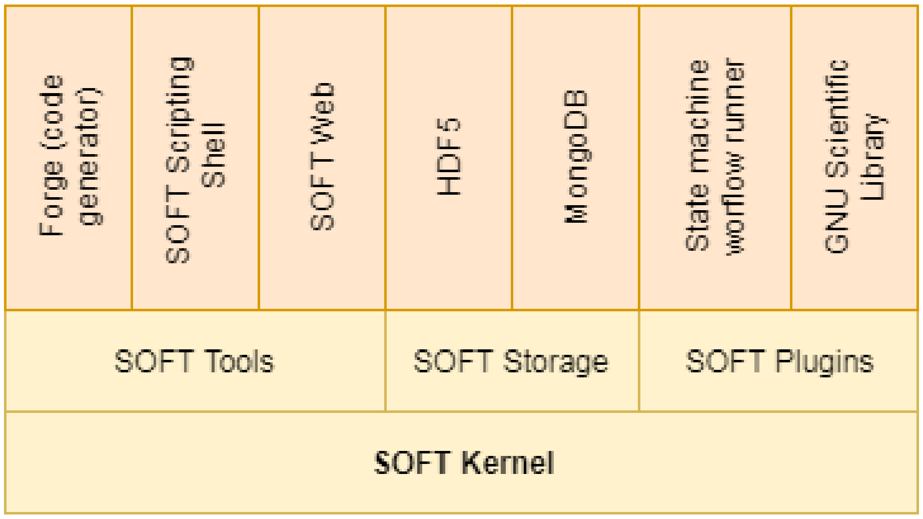


Figure 19: An overview of the SOFT modules. The SOFT Tools includes a framework for creating metadata-based code generators. Different storage drivers are supported. The idea is to extend these with interfaces to CKAN-services

Entities

An entity can be a single thing or object that represents something physical or nonphysical, concretely or abstract. The entity contains information about the data that constitutes the state of thing it describes. The entity does not contain the actual data, but describes what the different data fields are, in terms of name, data types, units, dimensionality etc. Information about data is often called meta data. Formal meta data enables for the correct interpretation of a set of data, which otherwise would be unreadable.

An example of an entity is ’Atom’, which can be defined as something that has a position, an atomic number (which characterizes the chemical element), mass, charge, etc. Another example of a completely different kind of entity can be a data reference-entity with properties such as name, description, license, access-url, media-type, format, etc). The first entity is suitable as an object in a simulation code, while the latter is more suitable for a data catalog distribution description (see dcat:Distribution). Entities allows for describing many aspects of the domain. While each entity describes a single unit of information, a collection of entities can describe the complete domain. See collections below.

Each published entity needs to be uniquely identified in order to avoid confusion. The entity identifier has therefore 3 separate elements: a name, a namespace and a version number. An entity named 'Particle' is unlikely to have the same meaning and the set of parameters across all domains. In particle physics, the entity 'Particle' would constitute matter and radiation, while in other fields the term 'Particle' can be a general term to describe something small. For this reason the SOFT5 entities have namespaces, similar to how vocabularies are defined in OWL. The version number is a pragmatic solution to handle how properties of an Entity might evolve during the development process. In order to handle different versions of a software, the entity version number can be used to identify the necessary transformation between two data sets.

Collections

A collection is defined as a set of instances of entities and relationships between them. Collections are themselves defined as entities, and can this contain other collections as well. This is useful to represent the knowledge of the domain where data exists, in order to find data that relates to other data, but also to uniquely identify a complete data set with a single identifier.

5.3 Translators

Translators provide the functionality to define the transformation step from one model to another. The interoperability framework provides the infrastructure to perform the actual data conversion. This allows for two software systems to work with two separate data models, but still exchange information. Off course, this assumes that the datamodels are semantically similar, i.e. they describe the same information, although using different representations.

5.4 Semantic Containers

The concepts that answers the third research question, "how can domain applications effectively utilize a wide range of open datasets published in different formats" is to define *semantic containers*. Semantic containers are sets of predefined standardized datamodels for specific domains.

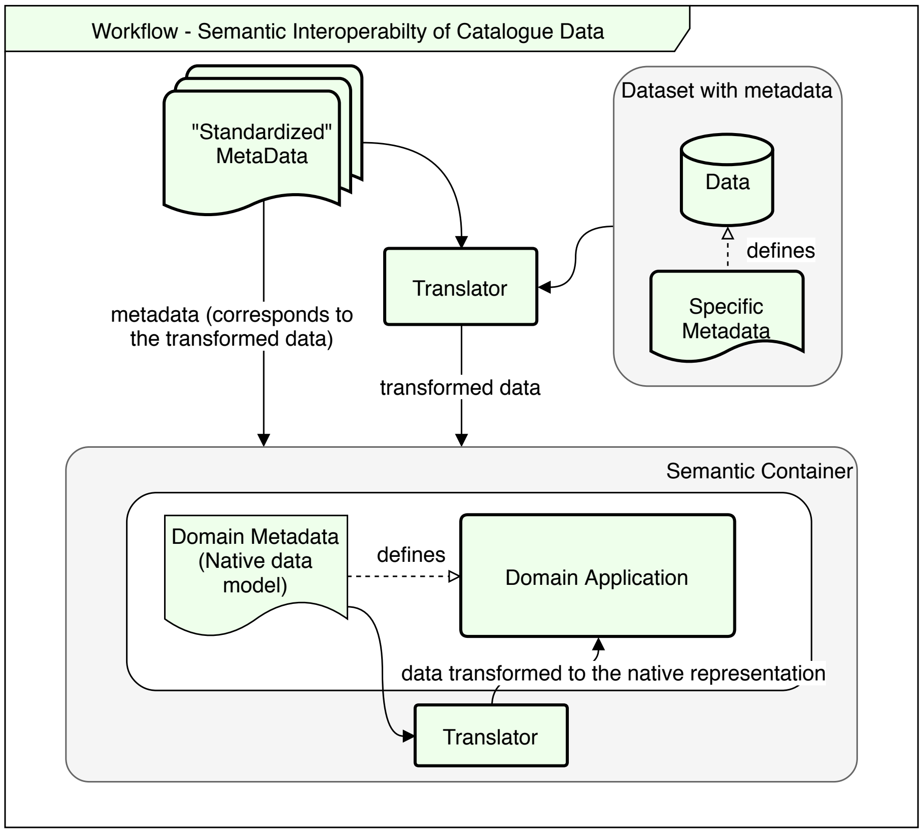

Workflow - Semantic Interoperabilty of Catalogue Data
"Standardized" MetaData
Dataset with metadata
Data
defines
Specific Metadata
Translator
metadata (corresponds to the transformed data)
transformed data
Semantic Container
Domain Metadata (Native data model)
defines
Domain Application
data transformed to the native representation
Translator

Figure 20: An application of semantic interoperability.

To allow semantic containers to be effective, the dataset catalogue services have to include information about which standard containers are semantically compatible with the provided datasets. The datasets themselves may be provided with specific metadata that can be translated to the standard container.

6 Ontology Based Semantic Framework Design

The framework for ontology based semantic classification and search is the research test-bed for the aforementioned ideas. The main design criteria are *modularity* (in order to be flexible and support replacing parts of the software), *extensibility* (in order to be able to easily add new software components). In addition the secondary criteria is *performance*. Although the implementation is a prototype for research purposes, it is essential that the user can experience relatively fast search responses.

6.1 Software Dependencies

The code is written in Python[1], and relies on several 3rd party libraries. The most essential libraries used are:

- **BeautifulSoup**[2] for cleaning up HTML markups
- **PyMongo**[3] for communicating with the MongoDB database backend
- **rdflib**[4] for managing RDF data
- **NLTK**[5] for Natural Language Processing
- **NumPy**[6] for computing with matrices
- **Flask**[7] for creating the search web site
- **Pandas**[8] for data manipulation and visualization
- **scikit-learn**[9] for clustering algorithms

6.2 Deployment and services

Figure 21 is a deployment diagram illustrating how the different framework software artifacts are physically located. In this section the different deployment services and artifacts are described.

[1]Python is a scripting language, managed by the Python Software Foundation `https://www.python.org/psf/`

[2]Beautiful Soup is a library for extracting information from HTML and XML documents. `https://www.crummy.com/software/BeautifulSoup/`

[3]MongoDB API for Python. `https://api.mongodb.com/python/current/`

[4]RDFLib is a Python library for working with RDF `https://github.com/RDFLib/rdflib`

[5]NLTK is a library for natural language processing`NLTKhttps://www.nltk.org/`

[6]NumPy supports scientific computing in Python `http://www.numpy.org/`

[7]Flask is a minimalistic web development framework for Python `http://flask.pocoo.org/`

[8]Pandas is a data analysis library `https://pandas.pydata.org/`

[9]scikit-learn is a library for machine learning in Python `http://scikit-learn.org/`

Web Application (Cloud Server)

The *Cloud Server* is a platform as a service (PaaS) that allows the search application to be accessible for anyone to use. At the time of writing the thesis, the service is running on Heroku[10] and can be reached at `ontosearch-heroku.herokuapp.com`. The Web Service is developed based on the Flask framework, which uses the semantic search module.

Storage server

Pre-computed indeces, tables, ontologies and other metadata are stored on the *Storage* server. MongoDB is used as a backend database, because it provides a flexible way of storing JSON[11] documents. Python dictionaries resemble JSON structures and converting back-and-forth between the two is trivial. To ensure good availability the database is hosted on mLab[12].

Workstation

The task of performing classification requires a lot of computational time. The main modules for running the classification are therefore offline operations. The results from the computation are uploaded to the storage service for further use by the search application. Ontologies and lexical databases resides locally on the workstation, but since datasets can be update externally they are fetched from the designated data catalogue service

Data Catalogue

The main artifacts for the entire framework are the datasets. A catalogue of available datasets with accompanying descriptions are managed by a CKAN server, hosted by SINTEF Digital.

[10]Heroku: Cloud Application Platform `https://www.heroku.com/`

[11]JavaScript Object Notation is a simple *keyword-value*-style text format used for data interchange. See `https:/www.json.org`

[12]mLab is a Database-as-a-Service for MongoDB. See `https://mlab.com/`

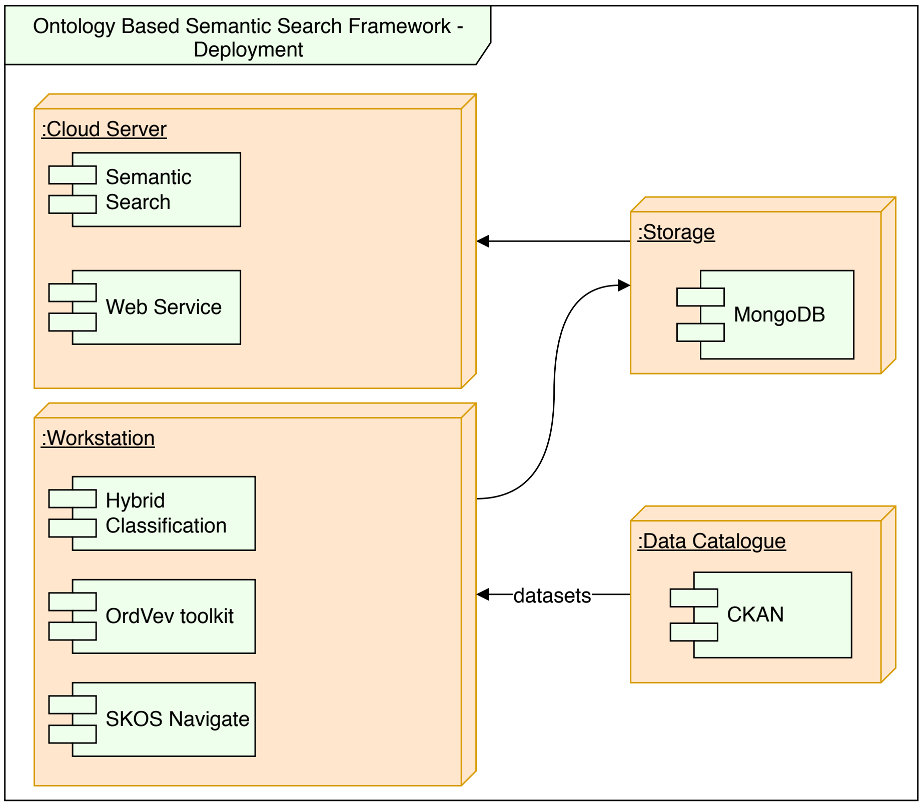


Figure 21: Deployment diagram

6.3 Development View

This section goes into details of describing the architecture from a development perspective. The modules described here are the semantic search, hybrid classification and OrdVev toolkit which are artifacts that can also be found in the deployment diagram (figure 21)

Hybrid Classification

Dataset classification (in this context) is the process of identifying the relevant scope of a dataset based on its metadata. The ontology defines the concepts. The task for the classification algorithm is to define links between datasets and concepts in the ontology. The links should describe the semantic relevance, i.e. indicate how strongly related the dataset is to a given concept. It is possible to have multiple links between concepts and datasets.

The output from the classifier is a set of similarity-relations using the OTD-similarity vocabulary defined in 4.1. The metadata is defined using the DCAT-AP vocabulary. This allows datasets to be tagged with information such as formats and standards.

In the case where there are known formats or standards that simplifies the classification process (by already knowing what the contents of the data must be, defined by the given format or standard), a specialized classifier can be implemented.

In order to make the design flexible for extension, a sensible design pattern] to use is the Decorator Design Pattern]. A design pattern is simply put a best-practise software design recipe for a set of well known problems. The decorator pattern adds the possibility to extend the behaviour of (in this case) the classifier, i.e. adding support for different types of classification algorithms. The final classifier is the constructed by linking together a set of specialized classifiers. This also opens up for the possibility to experiment with different types of classifiers.

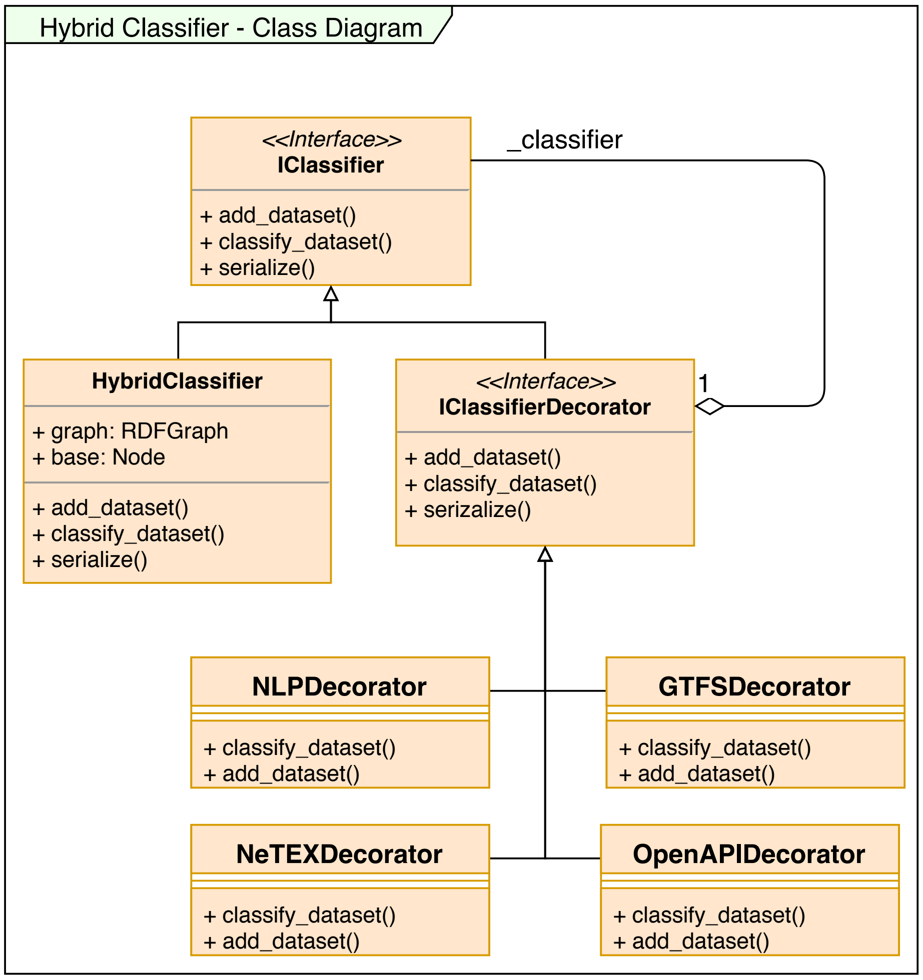


Figure 22: Class Diagram - Hybrid Classification

By comparing different compositions of classifiers, the differences can be measured. Figure 22 shows a class diagram that illustrates the main idea of the design. New classifiers must derive from the *IClassifierDecorator* and implement the appropriate interfaces.

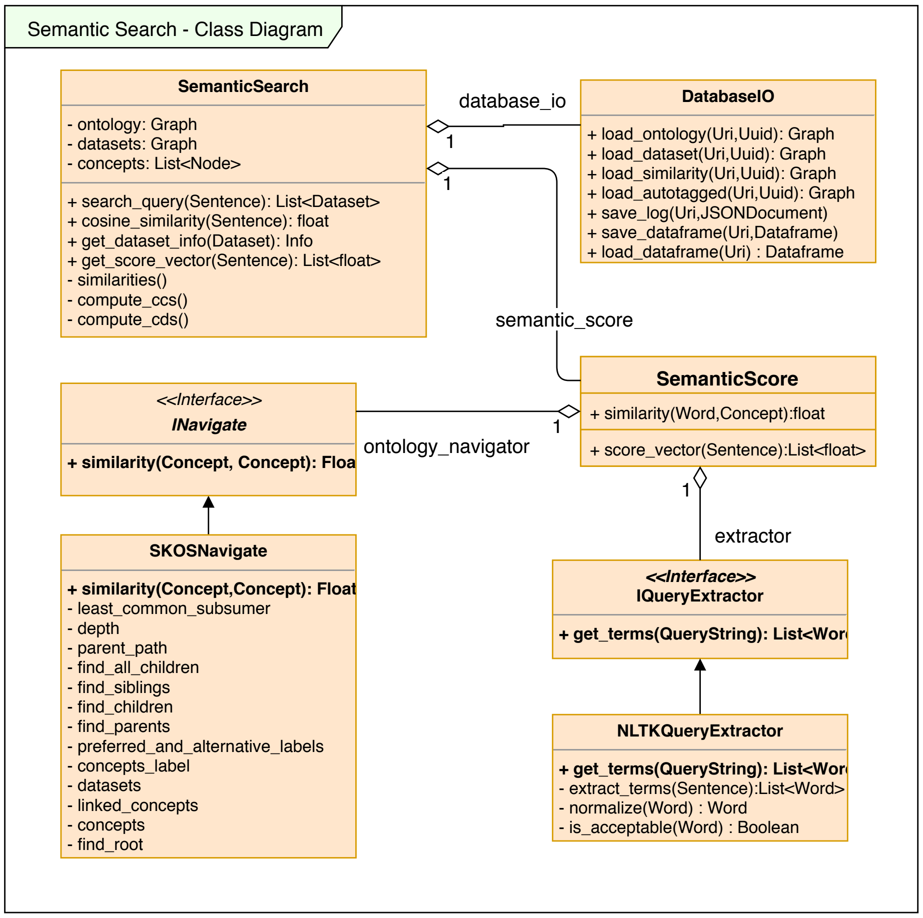

Semantic Search - Class Diagram
SemanticSearch
- ontology: Graph
- datasets: Graph
- concepts: List<Node>
+ search_query(Sentence): List<Dataset>
+ cosine_similarity(Sentence): float
+ get_dataset_info(Dataset): Info
+ get_score_vector(Sentence): List<float>
- similarities()
- compute_ccs()
- compute_cds()
database_io
1
1
DatabaseIO
+ load_ontology(Uri,Uuid): Graph
+ load_dataset(Uri,Uuid): Graph
+ load_similarity(Uri,Uuid): Graph
+ load_autotagged(Uri,Uuid): Graph
+ save_log(Uri,JSONDocument)
+ save_dataframe(Uri,Dataframe)
+ load_dataframe(Uri) : Dataframe
semantic_score
<<Interface>>
INavigate
+ similarity(Concept, Concept): Floa
ontology_navigator
1
SemanticScore
+ similarity(Word,Concept):float
+ score_vector(Sentence):List<float>
1
extractor
SKOSNavigate
+ similarity(Concept,Concept): Float
- least_common_subsumer
- depth
- parent_path
- find_all_children
- find_siblings
- find_children
- find_parents
- preferred_and_alternative_labels
- concepts_label
- datasets
- linked_concepts
- concepts
- find_root
<<Interface>>
IQueryExtractor
+ get_terms(QueryString): List<Wor
NLTKQueryExtractor
+ get_terms(QueryString): List<Wor
- extract_terms(Sentence):List<Word>
- normalize(Word) : Word
- is_acceptable(Word) : Boolean

Figure 23: Class Diagram - Semantic Search

Semantic Search

The semantic search module is primarily used on the search web application. In Figure 23 the main class *SemanticSearch* lists the main algorithms used:

- search_query is the main function which takes a search query and returns a list of filtered and ranked results.
- cosine_similarity is the main concept vector comparison algorithm
- compute_ccs computes the concept-concept relations from the ontology
- compute_cds computes the concept_dataset relations from the similarities computed by the hybrid classification module

In order to create a responsive search engine, the SemanticSeach class can perform the necessary computations the CCS and CDS matrices in an offline process and store the precomputed data in a database. The DatabaseIO class has a set of functions for storing and retrieving this data. When running an instance of the SemanticSearch class as a web application, the computations are skipped, and the precomputed data is fetched from the database server instead. The time it takes to perform this operation is hardly noticeable[13].

The most computationally intensive feature of the search engine is computing the concept vectors based on the search query. The *SemanticScore* class uses two interfaces *INavigate* and *IQueryExtractor* to perform this operation. Since different ontologies can be used, different strategies for navigating the ontology must be implemented. In the current prototype, a generic ontology navigator for SKOS-based ontologies is implemented.

Navigating an ontology is finding parent and child nodes and compute distances between concepts.

For extracting terms from a user query the natural language toolkit has been employed. This class can be changed with any other method or toolkit when needed.

This design gives flexibility in experimenting with different ontologies and NLP techniques, and allows for being deployed to a production server and run using pre-computed data for increase performance.

OrdVev

Unline WordNet, the Norwegian lexical database OrdVev has software library support. OrdVev is a set of RDF files that describes words, synsets (groups of synonyms) and relationship between synsets in terms of *hyponyms* (words that are defined as specialization of a more generic term. For example a *car* is a hyponym of*vehicle*), and *hypernyms* (the oppsite of hyponyms, namely the term that is more general, for example a *car* is a hypernym of a *SUV*). The hypernyms and hyponym resembles very much the parent-child (or is-a) relations used previously to define the the OTD ontology. It is therefore straight forward to implement support for navigating OrdVev the same way the as for the SKOS ontology. The most important feature of the OrdVev class is to be able to compute the similarity between two synsets. Figure 24 is a class diagram illustrating the OrdVev class. There are four artifacts that the OrdVev class rely on: wordgraph, wordsenses, synsets and hyponymOf_taxonomic.

[13]If the web application has been temporarily suspended due to inactivity, longer response time is expected

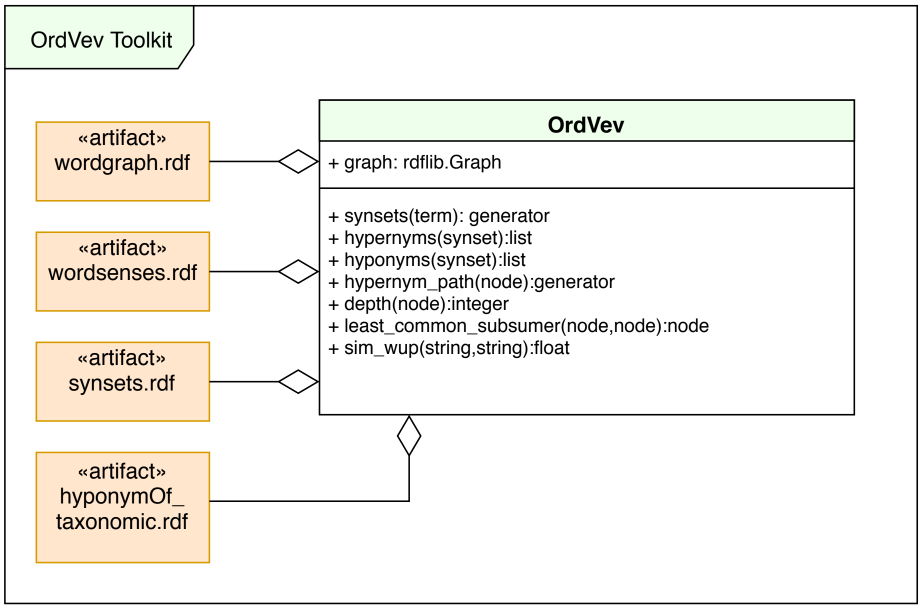
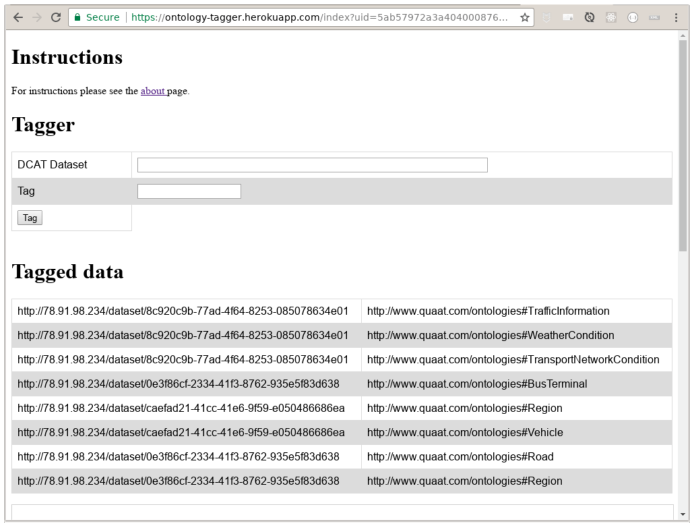

OrdVev Toolkit
«artifact» wordgraph.rdf
«artifact» wordsenses.rdf
«artifact» synsets.rdf
«artifact» hyponymOf_ taxonomic.rdf
OrdVev
+ graph: rdflib.Graph
+ synsets(term): generator
+ hypernyms(synset):list
+ hyponyms(synset):list
+ hypernym_path(node):generator
+ depth(node):integer
+ least_common_subsumer(node,node):node
+ sim_wup(string,string):float

Figure 24: Class Diagram - OrdVev

6.4 Web Application for manual tagging

As part of the research toolkit suite, a web application for performing manual tagging of datasets was implemented and deployed as a cloud service. The web application was originally intended for generating "live" similarity scores for use in the semantic search application. However, as the system became increasingly complex and resource demanding, the dataset tagging information was processed offline for produce the CDS matrixes


Secure | https://ontology-tagger.herokuapp.com/index?uid=5ab57972a3a404000876...

Instructions

For instructions please see the about page.

Tagger

DCAT Dataset	
Tag	
Tag	

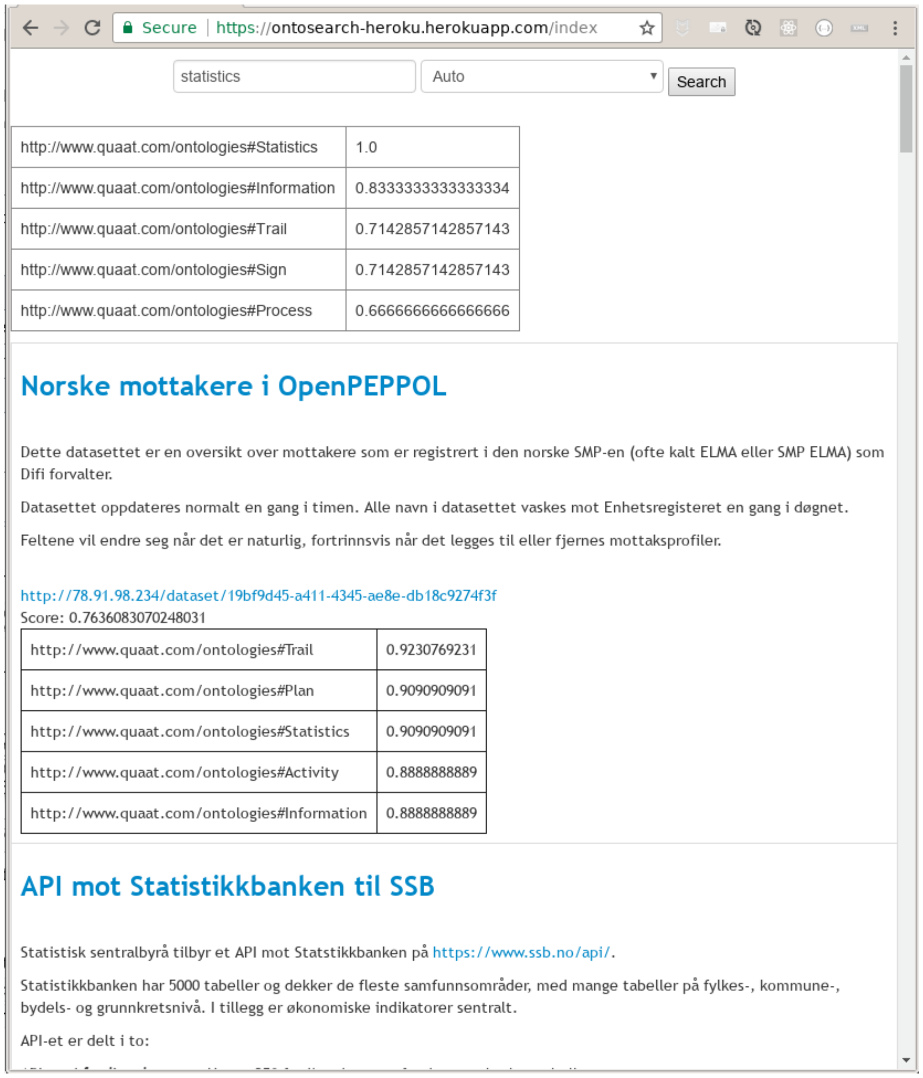
Tagged data

http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-085078634e01	http://www.quaat.com/ontologies#TrafficInformation
http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-085078634e01	http://www.quaat.com/ontologies#WeatherCondition
http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-085078634e01	http://www.quaat.com/ontologies#TransportNetworkCondition
http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f83d638	http://www.quaat.com/ontologies#BusTerminal
http://78.91.98.234/dataset/caefad21-41cc-41e6-9f59-e050486686ea	http://www.quaat.com/ontologies#Region
http://78.91.98.234/dataset/caefad21-41cc-41e6-9f59-e050486686ea	http://www.quaat.com/ontologies#Vehicle
http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f83d638	http://www.quaat.com/ontologies#Road
http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f83d638	http://www.quaat.com/ontologies#Region

Figure 25: Screenshot of the manual ontology tagger utility

6.5 Web-Based Search Application

The main user interface of the web application is a text input for entering the search query, and a button to push to start the search. In addition there is a drop-down list with options for which set of similarity data to use. The similarity data from the manual tagging contains the controlled semantic associations. To use this option, the drop-down selection *Tagged* must be selected. The other option is the results from the classification algorithm. To use the automatically generated similarities, the *Auto* options must be selected. A screenshot of the web application can be seen in figure 26.



Secure | https://ontosearch-heroku.herokuapp.com/index

statistics | Auto | Search

http://www.quaat.com/ontologies#Statistics	1.0
http://www.quaat.com/ontologies#Information	0.8333333333333334
http://www.quaat.com/ontologies#Trail	0.7142857142857143
http://www.quaat.com/ontologies#Sign	0.7142857142857143
http://www.quaat.com/ontologies#Process	0.6666666666666666

Norske mottakere i OpenPEPPOL

Dette datasettet er en oversikt over mottakere som er registrert i den norske SMP-en (ofte kalt ELMA eller SMP ELMA) som Difi forvalter.

Datasettet oppdateres normalt en gang i timen. Alle navn i datasettet vaskes mot Enhetsregisteret en gang i døgnet.

Feltene vil endre seg når det er naturlig, fortrinnsvis når det legges til eller fjernes mottaksprofiler.

http://78.91.98.234/dataset/19bf9d45-a411-4345-ae8e-db18c9274f3f
Score: 0.7636083070248031

http://www.quaat.com/ontologies#Trail	0.9230769231
http://www.quaat.com/ontologies#Plan	0.9090909091
http://www.quaat.com/ontologies#Statistics	0.9090909091
http://www.quaat.com/ontologies#Activity	0.8888888889
http://www.quaat.com/ontologies#Information	0.8888888889

API mot Statistikkbanken til SSB

Statistisk sentralbyrå tilbyr et API mot Statstikkbanken på https://www.ssb.no/api/.

Statistikkbanken har 5000 tabeller og dekker de fleste samfunnsområder, med mange tabeller på fylkes-, kommune-, bydels- og grunnkretsnivå. I tillegg er økonomiske indikatorer sentralt.

API-et er delt i to:

Figure 26: Screenshot of the ontology-based semantic search web application

The tables shown in the screenshot (under the search query and for each of the results) is feedback for use in the interpretation and evaluation of the search system. In the left column, the names of ontology concepts are shown. In the right column, the semantic score factor is shown. In the screenshot shown in figure 26 it can be observed that the term *statistics* correctly corresponded to the concept *OTD.Statistics* in the ontology. Under each search result the dataset-concept scores is shown.

7 Evaluation

The suitability of the ontology, the evaluation of the classification and search algorithm was tested in the following. The ontology was initially tested for completeness and compliance with the standards in chapter 7.1 The evaluation of the classification and search algorithm is presented in chapter 7.2. A brief discussion about the semantic interoperability is found in chapter 7.3.

7.1 Ontology Evaluation (RQ1)

Automatic Ontology verification

In order to formally validate the transport ontology against the SKOS standard, the SKOS testing tool[1] was employed. The SKOS testing tool is an online service for assessing the quality of SKOS vocabularies. The SKOS testing tool[2] is based on the qSKOS[37] quality assessment tool. The ontology was validated against a set of quality measurements. The quality measurements and results from this evaluation are presented in Table 7.1

[1]The SKOS testing tool is a open source project hosted on github https://github.com/sparna-git/skos-testing-tool

[2]http://labs.sparna.fr/skos-testing-tool/

Rule	**Result**
Ambiguous Notation References	OK
Cyclic Hierarchical Relations	OK
Disconnected Concept Clusters	OK
Disjoint Labels Violation	OK
Empty Labels	OK
Hierarchical Redundancy	OK
HTTP URI Scheme Violation	OK
Incomplete Language Coverage	OK
Inconsistent Preferred Labels	OK
Missing Labels	OK
Mapping Relations Misuse	OK
No Common Languages	OK
Orphan Concepts	OK
Omitted or Invalid Language Tags	OK
Overlapping Labels	OK
Omitted Top Concepts	OK
Relation Clashes	OK
Reflexively Related Concepts	OK
Solely Transitively Related Concepts	OK
Top Concepts Having Broader Concepts	OK
Undocumented Concepts	WARNING (107)
Unprintable Characters in Labels	OK
Unidirectionally Related Concepts	WARNING (109)
Undefined SKOS Resources	OK
Valueless Associative Relations	OK

Table 3: Results from the running the online SKOS testing tool

The analysis produced two warnings. The *Unidirectionally Related Concepts warning* is issued, claiming that the ontology has not included any reciprocal relations. The ontology is, however, defining broader/narrower relations, as well as the concept *Entity* as the top concept in the define scheme.

The second warning *Undocumented Concepts* is related to documentation in human-readable form through SKOS-properties such as SKOS.note, SKOS.definition and SKOS.example. As the ontology is currently a prototype for testing, and not meant to be published as a recommendation, it is an acceptable omission.

Testing for lexical coverage

The concepts defined in the transport ontology were labeled using as set of terms that describe the context. The terms that describe the concepts were compared with terms extracted from dataset descriptions, or from search queries. In order to compute how similar the terms are, a word similarity comparison was performed. The Python toolkit NLTK has support for WordNet and similarity functions, and in chapter 6.3 the supporting tools are built around the Norwegian OrdVev. In order for the comparison to make sense, the terms that are being compared must exist in the WordNet or OrdVev dictionary. A potential problem is that phrases that are too domain specific are not part of these dictionaries. To test this, the concept labels were looked up in both WordNet and OrdVev. The concepts were labeled using preferred labels (pref-labels) and alternatives labels (alt-labels). Note that there is no distinction between the two in the framework implementation. In Table 7.1 the number of labels and total coverage is recorded. The most important factor to look at is the total coverage of english and norwegian concepts.

Label type	Number	Coverage
English pref-label	67	65.7%
English alt-labels	4	3.9%
Norwegian pref-label	72	70.6%
Norwegian alt-label	9	8.8%
Total english concepts covered	71	69.6%
Total norwegian concepts covered	75	73.5%

Table 4: Dictionary coverage of english and norwegian concept labels

travel plan, personal transport, topographic point, public transport,traffic condition, point of interest, geographical information,weather forecast, street sign, service area, real time, bus stop,emission information, charging station, traffic information,environment information, route plan, traffic flow, railway junction,hydrometeorological information, bus terminal, air quality, traffic queue, park and ride, weather condition, meteorological information,traffic circle, transport mode, API description, transfer node,transport network condition, tracking information, travel information,t junction, tool booth,t junction, tool booth,bus station, coaching inn, transfer point, 3-way junction

Table 5: A collection of English terms used as concept labels in the ontology, but is missing from WordNet

landevei, persolig transport, topografisk punkt, offentlig transport, trafikkforhold, interessepunkt, geografisk informasjon, trafikkskilt, serviceområde, sanntid, ladestasjon, trafikkinformasjon, miljøinformasjuton, ruteplan, trafikkflyt, hydrometeorologisk informasjon, entitet, skysstasjon, meteorologisk informasjon, skinner, rundkjøring, transporttype, overføringspunkt, transportnettforhold, sporing, prediksjon, reiseinformation, 3-veis kryss, bomstasjon, trasse, skyss, geoinformasjon, discharge, utslippsdata, lading

Table 6: A collection of Norwegian terms used as concept labels, but is missing from OrdNet

In Table 5 and 6 the list of words that are missing in respectively WordNet and OrdVev dictionaries are listed. As a consequence of this, the computed similarity scores will be too low for these concepts.

7.2 Dataset Classification and Search Evaluation (RQ2)

Chapter overview

In this section, the classification and search performance is presented. The performance test was conducted using the developed semantic framework. A similar test was also conducted using CKAN, as a representative platform for the state of the art. The test dataset used to measure the performance of the classification and search algorithms is accessible via the SINTEF Digital CKAN server (7.2.1)

In preparation for the evaluation, every dataset relevant to the transport domain was manually tagged to one or more concepts in the ontology. This data has been compiled into a document that describes the datasets and its relevant concepts. The document is one of the contributions to this project[38]

The measuring methods defined in (7.2.3) were chosen to give good indicators of the search performance.

7.2.1 Retrieving the datasets

The datasets used in the evaluation were fetched from the SINTEF Digital CKAN server. At the time of writing the report, the site was running on `http://78.91.98.234:5000/`. The datasets

The datasets can be retrieved using the CKAN Rest API.

```
$ curl -O http://78.91.98.234:5000/api/3/action/package_list
```

The DCAT-AP metadata describing each dataset can be retrieved:

```
$ http://78.91.98.234:5000/api/3/action/package_show?id=<name>
```

7.2.2 Manually tagged datasets - Standard Evaluation Benchmark

In order to prepare a benchmark to measure the quality of classification and search against, the relevant transport datasets found in 7.2.1 were manually tagged with the appropriate ontology concept. The manual tagging has been compiled in a document[38] (Figure 34).

Appendix E shows how the manually tagged datasets were used to generate the similarity scores. The resulting similarity graph can be found in the OTD semantic framework repository on github[3]

7.2.3 Measuring methods

To get an objective indication of how well the search engine performs, a set of performance metrics were established. By using the reference standard evaluation benchmark 7.2.2 it is possible to count the number of retrieved datasets that are

[3]The concept-dataset links with similarity scores generated from the manual tagging is available on `https://github.com/quaat/OTD-semantic-framework/blob/master/rdf/tagged-v0.9.json`

relevant (true positives) and which are not relevant (false positives). In addition it is possible to count the number of relevant documents not retrieved (false negatives) as well as irrelevant documents not retrieved (true negatives).

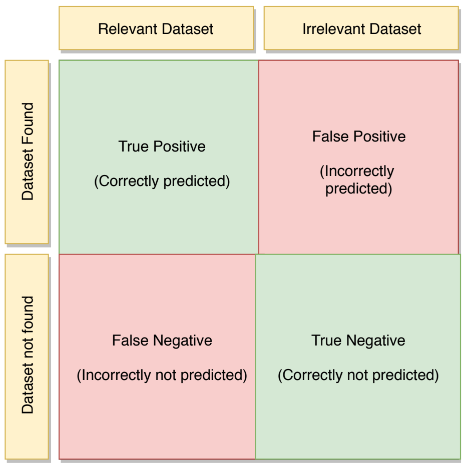


Figure 27: Table used for measuring performance of the search system. The table compares the true case (actual) with the predictions. In the field of machine learning, this table is known as a *confusion matrix*

The matrix depicted in figure 27 can be used to count the true positives T_P, false negatives F_N, false positives F_P and true negatives T_N. From this it is possible to derive a set of performance indicators:

- **Precision** [39] indicates how relevant the results. It does not, however, give an indication of completeness (that all relevant results were found) Precision Pr can be calculated from the confusion matrix as $Pr = \frac{T_P}{T_P+F_P}$
- **Recall** [39] is an indication of completeness, i.e. the fraction of relevant datasets that were found. Recall is not affected by irrelevant results, and thus cannot guarantee that a specific result is relevant. Recall Rc can be calculated as $Rc = \frac{T_P}{T_P+F_N}$
- F_1 **score** or F-measure[40]. Precision and recall presents two very different perspectives that are equally important. The F_1-score unifies these two indicators by computing the harmonic mean: $F_1 = (\frac{Rc^{-1}+Pr^{-1}}{2})^{-1} = 2\frac{Pr \cdot Rc}{Pr+Rc}$.
- **Accuracy** measures the fraction of true positives (correctly predicted) and true negatives (correctly not predicted) (the green squares of the confusion matrix) among all datasets. Accuracy Acc is measured by $Acc = \frac{T_P+T_N}{T_P+T_N+F_P+F_N}$

7.2.4 Comparing the similarity of the English and Norwegian Concept Labels

This section is relevant for answering the research question RQ 2.1 "*How can language independence be supported?*" One of the preconditions for supporting language independence is that all concepts have labels with words that can be found in a dictionary. In chapter 7.1, the lexical coverage was investigated. In this section, the similarity between concept labels for all concepts are compared using both the English and Norwegian dictionaries. Appendix C includes the Python source code for performing the comparison.

The comparison steps are the following;

1. For each concept C_i in the ontology, compare the concept labels with all other concepts C_j using OrdVev $S_{OrdVev,i,j}$ and WordNet $S_{WordNet,i,j}$.
2. Compute the relative distance between the similarity scores from using WordNet $S_{WordNet,i,j}$ and OrdVev $S_{OrdVev,i,j}$: $d_{i,j} = |S_{WordNet,i,j} - S_{OrdVev,i,j}|$

Figure 28 indicates the difference in calculating the CCS table using OrdVev versus using WordNet. Each pixel in the plot is an individual concept-concept similarity score difference. Yellow pixels indicate 100% difference, which is an indication these are concepts where the label has no valid definition in the dictionary. Green pixels indicates a significant difference in the interpretation of semantic similarity between two concepts. Ideally the entire plot should have been dark blue, which would indicate that the semantic differences between the concepts would be the same for both languages.

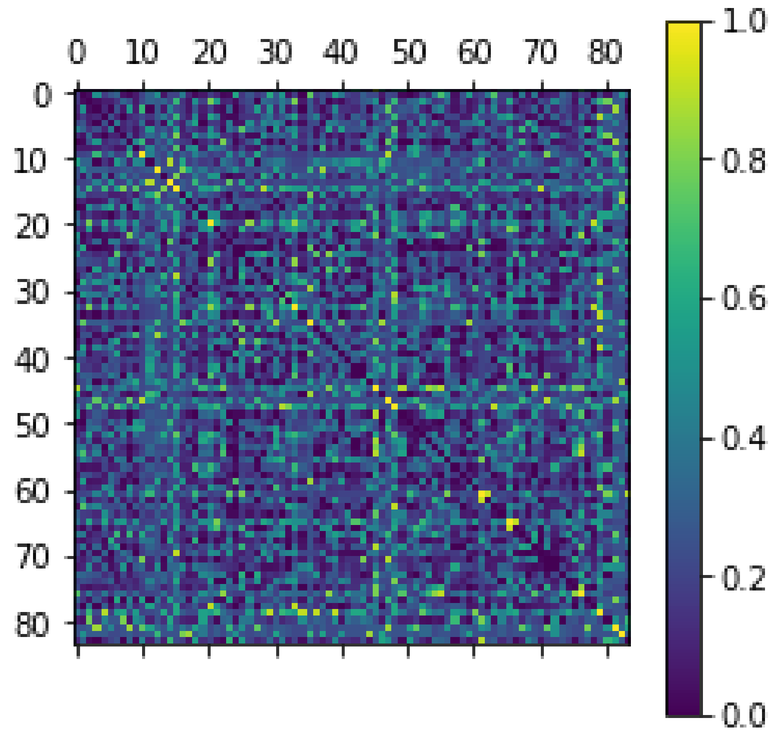
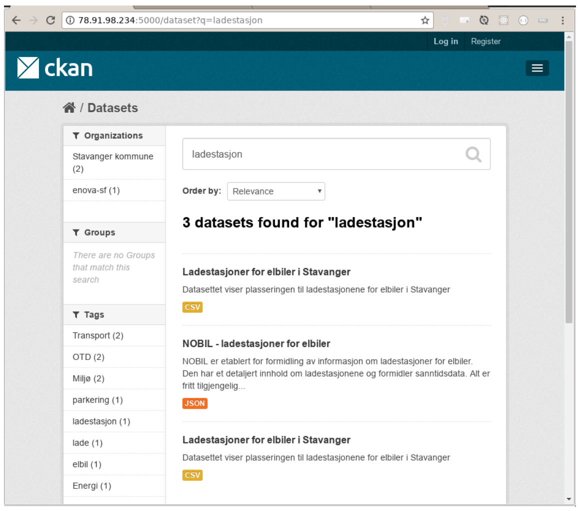


Figure 28: The plot compares the difference in label similarity between Norwegian and English labels, for all concepts in the ontology. Dark blue indicates similar scores, while the lighter green and yellow show a significant difference. Ideally the entire plot should have been dark blue, which would indicate that the semantic differences between the concepts would be the same for both languages

7.2.5 CKAN Comparison

CKAN uses a keyword based search system. Dataset descriptions that match the same terms or phrases as the search query will be presented in the search results. If the dataset descriptions does not match the exact words used in the query (i.e. if synonyms are used in the search instead), the dataset will not be part of the results.

Example 7.2.1 *Searching 'ladestasjon' (charging station) using CKAN*



78.91.98.234:5000/dataset?q=ladestasjon

Log in Register

ckan

/ Datasets

Organizations
- Stavanger kommune (2)
- enova-sf (1)

Groups

There are no Groups that match this search

Tags
- Transport (2)
- OTD (2)
- Miljø (2)
- parkering (1)
- ladestasjon (1)
- lade (1)
- elbil (1)
- Energi (1)

ladestasjon

Order by: Relevance

3 datasets found for "ladestasjon"

Ladestasjoner for elbiler i Stavanger

Datasettet viser plasseringen til ladestasjonene for elbiler i Stavanger

CSV

NOBIL - ladestasjoner for elbiler

NOBIL er etablert for formidling av informasjon om ladestasjoner for elbiler. Den har et detaljert innhold om ladestasjonene og formidler sanntidsdata. Alt er fritt tilgjengelig...

JSON

Ladestasjoner for elbiler i Stavanger

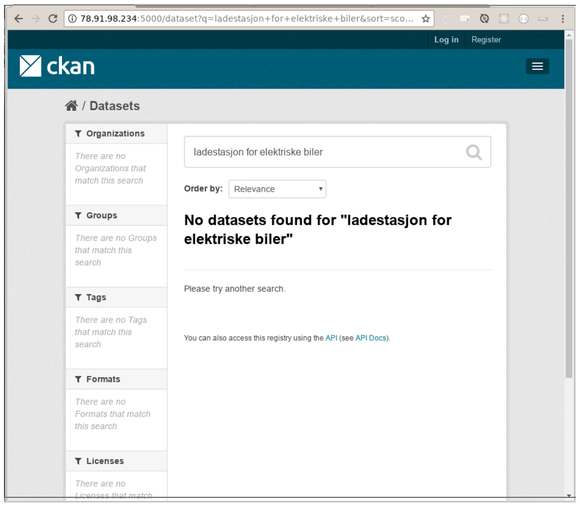
Datasettet viser plasseringen til ladestasjonene for elbiler i Stavanger

CSV

Figure 29: Searching for the term *ladestasjon* (charging station) using the CKAN search

In Example 29, CKAN is able to find the correct datasets. This is due to the fact that "ladestasjon" (charging station) is a word in the dataset descriptions. If the search is extended to "ladestasjon for elektriske biler" (charging station for electrical cars), as shown in Example 30, the search engine fails to return anything, even if *some* of the words fit with the dataset description.

Example 7.2.2 *Extending the search to ladestasjon of elektriske biler (charging stations for electrical cars)*



78.91.98.234:5000/dataset?q=ladestasjon+for+elektriske+biler&sort=sco...

Log in Register

ckan

/ Datasets

Organizations

There are no Organizations that match this search

Groups

There are no Groups that match this search

Tags

There are no Tags that match this search

Formats

There are no Formats that match this search

Licenses

There are no Licenses that match

ladestasjon for elektriske biler

Order by: Relevance

No datasets found for "ladestasjon for elektriske biler"

Please try another search.

You can also access this registry using the API (see API Docs).

Figure 30: Searching for the term *ladestasjon for elektriske biler* (charging station for electrical cars) using the CKAN search

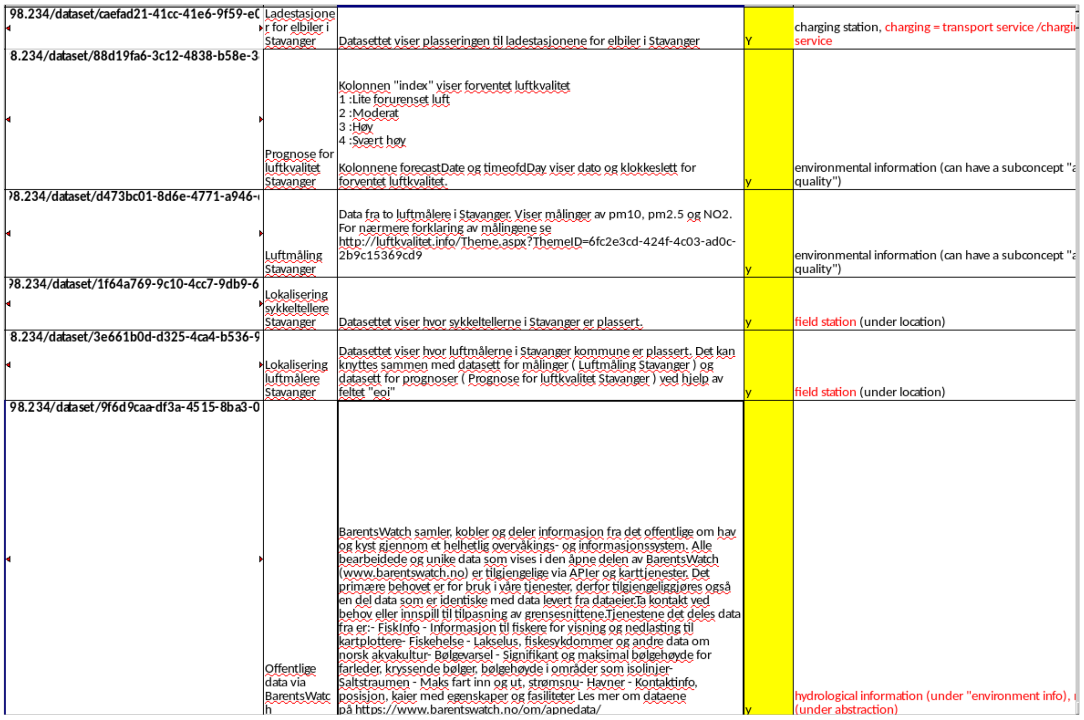
7.2.6 Evaluating Search Results

In the ontology-based semantic search application developed in the project (see screen-shot in figure 26), the user may enter a search string and select whether the results should be drawn from the set of manually tagged dataset, or from the automatically tagged datasets. This functionality is added to be able to compare the differences between what is manually tagged, and what has been automatically classified. Note that the manually tagged datasets are fewer than the automatically tagged. A direct side-by-side comparison of the ranked results is therefore not advisable.

Based on the document for manually tagged datasets (Figure 31), a set of search phrases was constructed. The tagged datasets only covers a limited set of concepts and the search phrases were limited to only 7 queries. The queries are listed in T able 7

Measuring CKAN search results

In order to measure the CKAN search results the search queries were manually fed into the CKAN search field. The results were checked against the manually tagged benchmark document to count the number of true and false positives and true and false negatives.



98.234/dataset/caefad21-41cc-41e6-9f59-e(	Ladestasjoner for elbiler i Stavanger	Datasettet viser plasseringen til ladestasjonene for elbiler i Stavanger	Y	charging station, charging = transport service /chargi service
8.234/dataset/88d19fa6-3c12-4838-b58e-3	Prognose for luftkvalitet Stavanger	Kolonnen "index" viser forventet luftkvalitet 1 :Lite forurenset luft 2 :Moderat 3 :Høy 4 :Svært høy Kolonnene forecastDate og timeofdDay viser dato og klokkeslett for forventet luftkvalitet.	y	environmental information (can have a subconcept "a quality")
98.234/dataset/d473bc01-8d6e-4771-a946-	Luftmåling Stavanger	Data fra to luftmålere i Stavanger. Viser målinger av pm10, pm2.5 og NO2. For nærmere forklaring av målingene se http://luftkvalitet.info/Theme.aspx?ThemeID=6fc2e3cd-424f-4c03-ad0c-2b9c15369cd9	y	environmental information (can have a subconcept "a quality")
98.234/dataset/1f64a769-9c10-4cc7-9db9-6	Lokalisering sykkeltellere Stavanger	Datasettet viser hvor sykkeltellerne i Stavanger er plassert.	y	field station (under location)
8.234/dataset/3e661b0d-d325-4ca4-b536-9	Lokalisering luftmålere Stavanger	Datasettet viser hvor luftmålerne i Stavanger kommune er plassert. Det kan knyttes sammen med datasett for målinger (Luftmåling Stavanger) og datasett for prognoser (Prognose for luftkvalitet Stavanger) ved hjelp av feltet "eoi"	y	field station (under location)
98.234/dataset/9f6d9caa-df3a-4515-8ba3-0	Offentlige data via BarentsWatch	BarentsWatch samler, kobler og deler informasjon fra det offentlige om hav og kyst gjennom et helhetlig overvåkings- og informasjonssystem. Alle bearbeidede og unike data som vises i den åpne delen av BarentsWatch (www.barentswatch.no) er tilgjengelige via APIer og karttjenester. Det primære behovet er for bruk i våre tjenester, derfor tilgjengeliggjøres også en del data som er identiske med data levert fra dataeier.Ta kontakt ved behov eller innspill til tilpasning av grensesnittene.Tjenestene det deles data fra er:- FiskInfo - Informasjon til fiskere for visning og nedlasting til kartplottere- Fiskehelse - Lakselus, fiskesykdommer og andre data om norsk akvakultur- Bølgevarsel - Signifikant og maksimal bølgehøyde for farleder, kryssende bølger, bølgehøyde i områder som isolinjer- Saltstraumen - Maks fart inn og ut, strømsnu- Havner - Kontaktinfo, posisjon, kaier med egenskaper og fasiliteter Les mer om dataene på https://www.barentswatch.no/om/apnedata/	y	hydrological information (under "environment info), (under abstraction)

Figure 31: The evaluation benchmark document which indicates which concepts are relevant for each dataset in the test-set

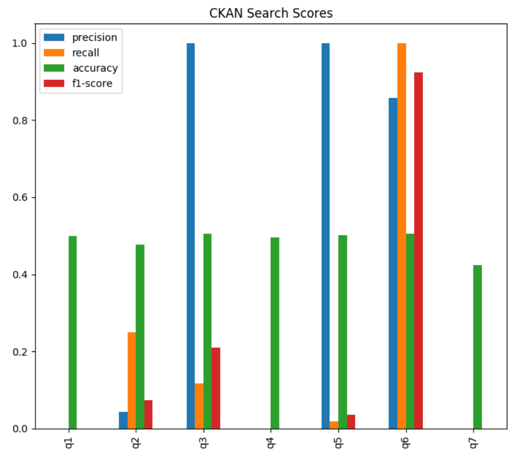
Name	Query
Q1	electric car charging station
Q2	information on air quality
Q3	location
Q4	statistics
Q5	trail
Q6	bike
Q6	map

Table 7: The search queries used for comparison with manually tagged benchmark results

Query	Number of datasets	Correct datasets	Incorrect datasets	Missing relevant datasets
Q1	0	0	0	1
Q2	23	1	22	3
Q3	4	4	0	30
Q4	3	0	3	5
Q5	1	1	0	54
Q6	7	6	1	0
Q7	80	0	80	3

Table 8: Record showing results from performing manual search with CKAN

The metrics precision, recall, accuracy and f1-score were computed based on the results from table8. Figure 32 shows how these metrics plotted as bar graphs for each individual query. The statistical measures; average, mean, standard deviation, variance, min and max, were computed based on the population of the 7 queries and are shown in Figure 33.



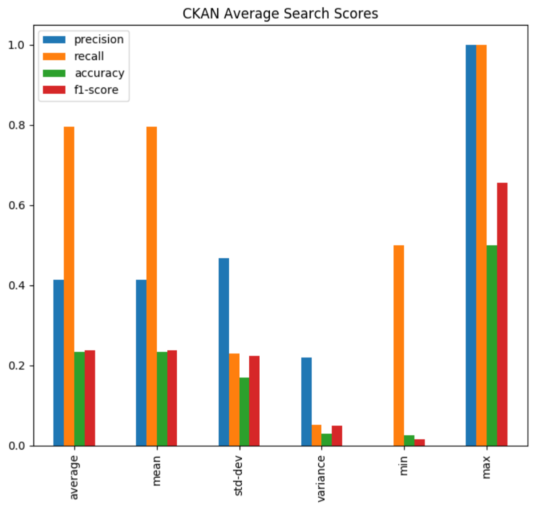


Figure 32: The chart compares the precision, recall, accuracy and f1-scores of 7 different queries

Figure 33: Search result statistics based on the 7 input queries

Initial search results

The initial search results from the ontology based semantic search engine are presented in the following. To measure the efficiency of the developed ontology-based semantic search engine, the same queries used to measure the CKAN search results (Table 6) were applied. The results are either based on the manually-tagged datasets or the automatically classified datasets.

The results were generated by first defining what the expected results should be, and comparing these with the actual search results. The results based from manually tagged dataset are shown in table 9. The results from the automatically classified datasets are shown in Table 10. The same results are presented visually as bar-charts in Figure 34 and 35

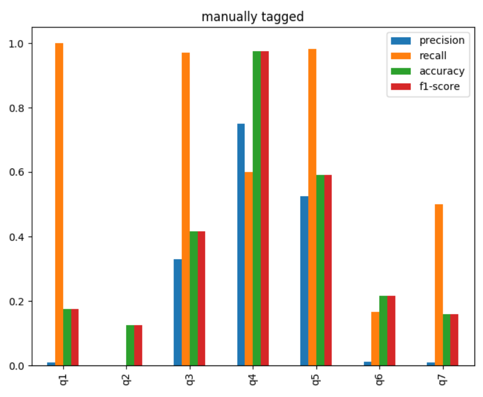


Figure 34: Manually tagged search results

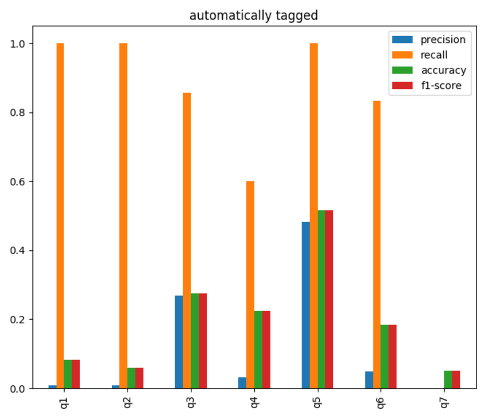


Figure 35: Automatically tagged search results

Query	Precision	Recall	Accuracy	f1-score
q1	0.010000	1.000000	0.175000	0.175000
q2	0.000000	0.000000	0.125000	0.125000
q3	0.330097	0.971429	0.416667	0.416667
q4	0.750000	0.600000	0.975000	0.975000
q5	0.524752	0.981481	0.591667	0.591667
q6	0.011111	0.166667	0.216667	0.216667
q7	0.009901	0.500000	0.158333	0.158333

Table 9: Search results based on the manually tagged dataset

Query	Precision	Recall	Accuracy	f1-score
q1	0.009009	1.000000	0.083333	0.083333
q2	0.008772	1.000000	0.058333	0.058333
q3	0.267857	0.857143	0.275000	0.275000
q4	0.031915	0.600000	0.225000	0.225000
q5	0.482143	1.000000	0.516667	0.516667
q6	0.049020	0.833333	0.183333	0.183333
q7	0.000000	0.000000	0.050000	0.050000

Table 10: Automatically tagged - search results

Measuring search results of the developed search engine

An advancement from the previous section is to look at adjustable parameters that has impact on the search quality. Concept relevance[4] is adjusted by using a semantic similarity score threshold value T_c. If a dataset has a similarity score higher than the threshold, the dataset is expected to part of the search result. If T_c is set to 0, all datasets would be defined as the relevant dataset. If T_c were set to 1, only those datasets that are tagged to a concept with a similarity scores = 1 would be the considered correct.

When the search query is submitted to the search engine, the resulting datasets will have a relevance score based on its similarity with the search query. The search engine will filter out all results that are below this threshold. The threshold value T_s is a parameter which decides which dataset should be a part of the search results. The effect of varying the values T_c and T_c are significant, and should be considered when interpreting the search results.

Different result types were compared; the results based on the manual tagging (manual), the automatic classification (auto) and a combination of the two (union). The Python class *SimMap* defined in Appendix G builds map of all datasets that are relevant for a specific concept, performs the search, and compares the search results with the expected results. The results are generated by running a set of automatic evaluations over a range of threshold factors T_c and T_s. The code and results can be seen in Appendix F

```
def gen_analysis(self, query, expected_tag, threshold, *e, **opts):
    search_res = set()
    for tag_type in e:
        res = self.ontology.search_query(query,
                                         tag_type, threshold)
```

[4]Concept relevance is a limit for how similar a concept and a dataset must be for a concept to be considered relevant

```
        for r in res:
            search_res.add(r[1][2])
cms = confusion_matrix_scores(search_res,
              self.set_map[expected_tag],
              self.all_datasets)
return [precision(cms),
        recall(cms),
        accuracy(cms),
        f1_score(cms)]
```

Listing 7.1: The gen_analysis method of the SimMap class

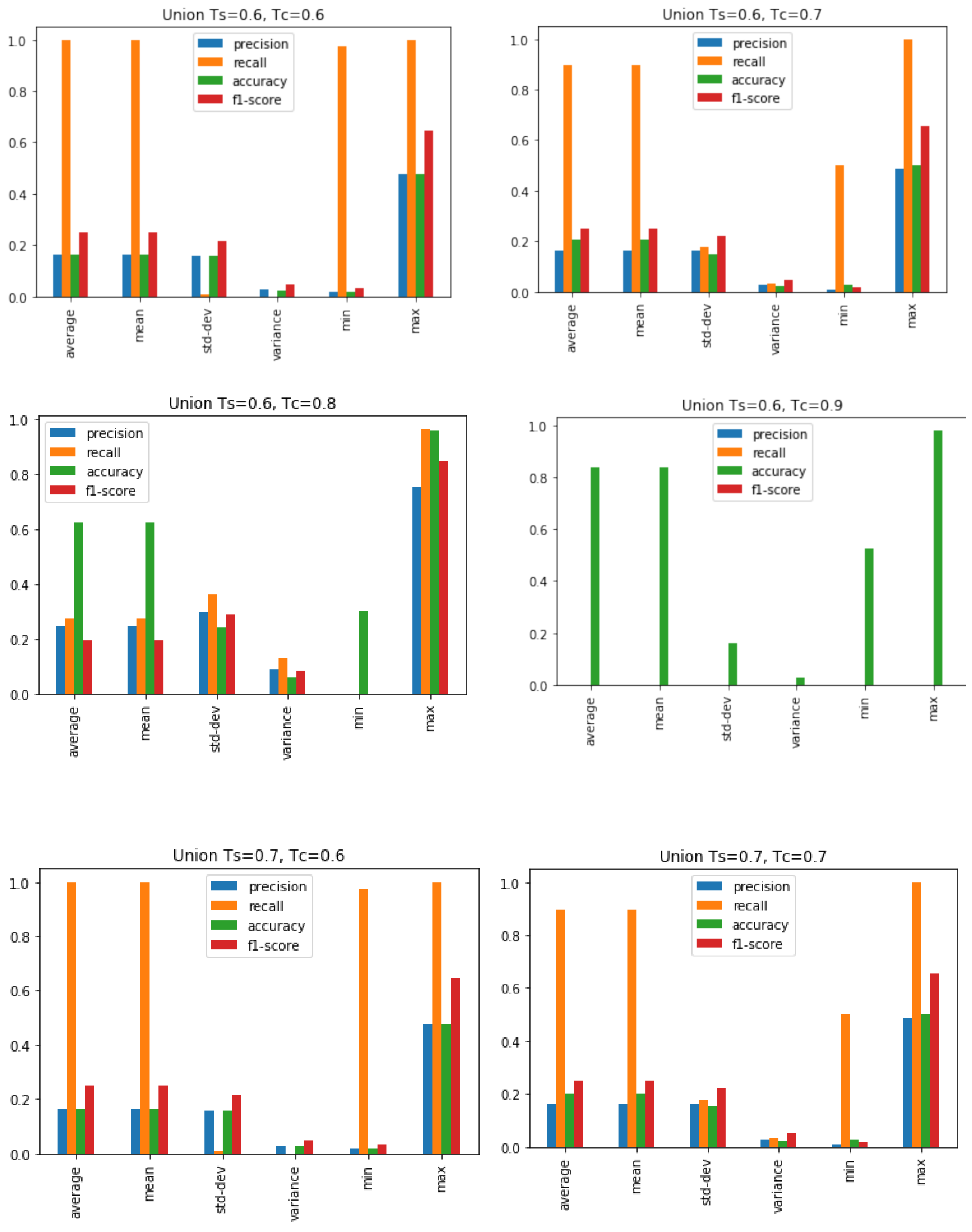


Figure 36: The summary results (1 of 2) from all queries combined using similarities from both the automatic and the manual tagging. The search result threshold (T_s) is varied from 0.6 to 0.7 with a concept threshold T_c varying between 0.6 and 0.9

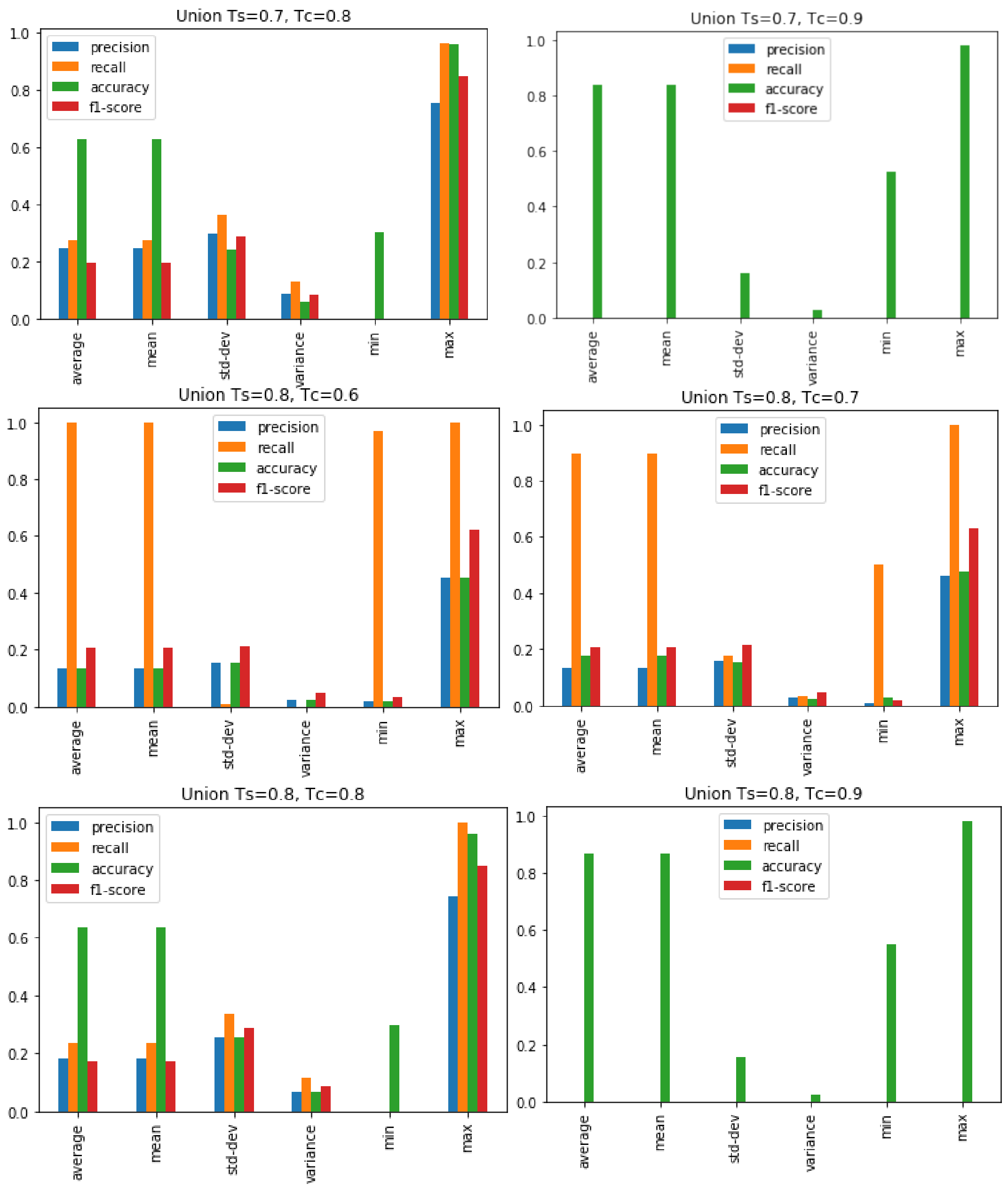


Figure 37: The summary results (2 of 2) from all queries combined using similarities from both the automatic and the manual tagging .The search result threshold (T_s) is varied from 0.7 to 0.8 with a concept threshold T_c varying between 0.6 and 0.9

The results from running a limited set of tests are shown in Figure 36 and 37. The semantic search engine outperforms the keyword based search engine, when the input query describes a similar, but not the same, phrases used to annotate the data. The search engine is very sensitive to the search and concept threshold values, and the optimal area seems to lie somewhere between 0.7 and 0.8.

7.3 Semantic Interoperability Evaluation (RQ3)

The development of the semantic similarity test framework has only reached an early design stage. As there are no prototype implementations, the idea has not been demonstrated in the scope of this project.

The of the conceptual design is illustrated in this figure:

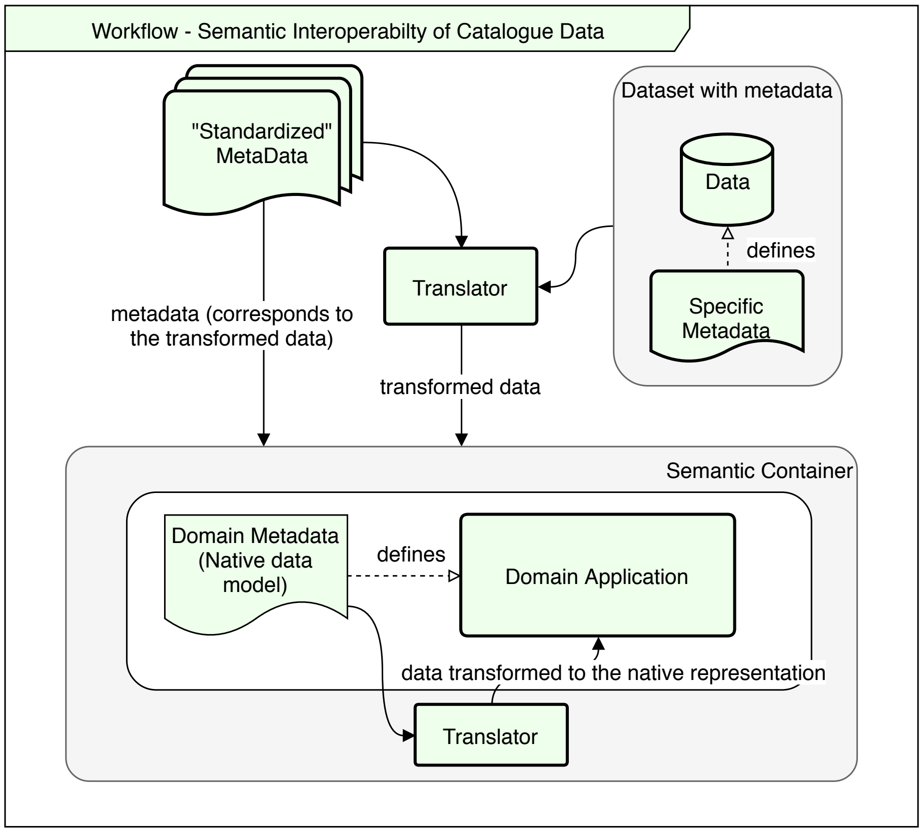

Workflow - Semantic Interoperabilty of Catalogue Data
"Standardized" MetaData
Dataset with metadata
Data
defines
Specific Metadata
Translator
metadata (corresponds to the transformed data)
transformed data
Semantic Container
Domain Metadata (Native data model)
defines
Domain Application
data transformed to the native representation
Translator

Figure 38: An application of semantic interoperability.

By linking dataset-specific metadata to the dataset description, a model-to-model translator can convert the data contents according a standardized published schema. The transformed data is then made available to a domain application that may or may not have its own metadata model defined (or it can be written to use the standard metadata directly). If the domain application is using its own data model, the translator is invoked again to adopt the data to a schema the domain application can read. When performing these steps, the domain application can read any available data from any data source as long as i)the dataset-specific metadata is semantically equivalent to a defined standard, ii) the semantic interoperability platform has support for reading the contents of the datafile and iii) the domain application is build to support interchanging data with the interoperability platform.

8 Discussion and Conclusion

8.1 Discussion

The aim of this project was to design and evaluate ontology based semantic technology that could improve the discoverability and utilization of open transport data. Although the focus has been on transport data, the methods discussed are general and could be applicable for many different domains. The scope of this project was very wide, as the project includes topics from ontology design, classification and semantic search (which involves natural language processing) and semantic interoperability. The main contributions from the project are; the design of an ontology that captures (a subset) of the transport domain, a framework for testing and evaluating ontology based classification and search and evaluation benchmark that can be used to measure the performance of different search engines or methodologies. The study additionally provides a proof-of-concept design and implementation of language independence in search and classification, and a design for allowing software applications to utilize information stored in federated datasets, without having to manage different file formats.

The first goal of the project was to develop an ontology that was suitable for covering the contexts of a wide range of datasets. The ontology should be appropriate for classification problems and ontology based semantic search. Existing ontologies were evaluated, but it was necessary to develop a new ontology with a strict hierarchical structure. To allow for any transport related dataset to fit in the context of a concept in the ontology, an upper level ontology strategy was adopted. An upper level ontology is a small and abstract set of concepts from which all other concept can be refined. Three categories for the upper level concepts were defined, namely: object, process and abstraction. These three categories were chosen based on the theory that these were unambiguous, not because they are well suited as categories to describe the transport domain. Future work could consider which set of abstract concepts that best fit the transport domain. By introducing very abstract concepts, and specializing these further down the classification-hierarchy, the first research question was addressed ("*How can the design of transport ontologies be improved to support the classification of current and future dataset?*"). The upper ontology allows for new concepts to be introduced without changing the relations of existing ones. The developed ontology was by no means complete and production ready, but it served its purpose for evaluating the classification and semantic search methods (also developed in this project).

One design choice was to use WordNet and OrdVev as lexical databases for defining the semantics of each concept. The problem with this approach is that terms used in the transport domain do not always fit the terms defined in a general dictionary. In future developments of the ontology one should consider how existing domain vocabularies could be included. The design of the ontology itself is a larger issue, as it requires close cooperation with domain experts. The problem with this is the perspective of the designer. It is easier to be pragmatic and define concepts and relations based on the problems to be solved, than to find *the* correct (real world) representation.

An important contribution to perform the evaluation steps was the development of the evaluation benchmark document, which includes a list of the datasets that are part of the test, and the related concepts for each dataset. The document was created in cooperation with domain experts at SINTEF Digital (Dr Marit K. Natvig and Dr. Shanshan Jiang). The reusability of the document could be improved by clearly referring the named concepts of the ontology instead of informal textual descriptions.

To answer the second research question, "*Can semantic technology be applied to improve the discoverabilty of open transportation datasets?*", different methods for dataset classification were evaluated. The idea was that when a dataset is correctly linked to a set of concepts, the dataset is found when the same set of concepts are identified as relevant in a specific context (for instance during search). In most cases, only the description of the datasets could be read. To challenge this problem, different strategies were applied. Manual tagging is a slow and labor intensive method that produces high quality relations between concepts and dataset. The alternative approach was to use known information about a set of standard file formats where the scope of the contents are well known. Making the correct link to concepts in these cases was trivial.

The automatic classification is a challenging problem which very much depends on the quantity and quality of the data. While some models are shown to produce good results in some cases, it is difficult to apply them in the generic cases. Supervised classification of datasets turned out to give poor results, due to too few dataset with very limited and/or poor textual descriptions. With a limited set of about 200 datasets it is unlikely that employing machine learning for automatic text classification would result in anything but an over-fitted model that cannot be reused. On the positive side, with few datasets, the manual tagging method is a viable option, and ensure good relations between the datasets and the concepts.

In order to extract the most significant words from the description text, the TF-IDF (term frequency - inverse document frequency) technique was tested. This technique is often used in text mining to identify the importance of the different terms in document. When applied to our dataset, however, key terms was weighted very low, due to the fact that some of these terms were only used once, and only in one document. On the positive side, the revised OTD ontology contains a set of significant words. In order to extract the most relevant terms, it is enough to compare the semantic similarity between the terms and the concept labels. Semantic similarity between words can be computed using techniques such as Wu & Palmer. For the Norwegian ordvev a similar method was implemented. The concept-concept-similarity (CCS) matrix is a table where the semantic similarity score between all concepts is defined. Because the ontology was defined as hierarchy of *is-a* relations, it was reasonable to use the Wu&Palmer similarity function on the classification tree in the ontology directly. Future work should include different similarity measures (especially hybrid methods which include edge-based and information-content based similarity measures)

The idea of applying a framework for semantic interoperability to allow software applications to utilize datasets without managing different file formats, defines a potentially powerful paradigm for future software applications. This topic has gotten the least attention in this project, overshadowed by the more fundamental problems of finding the datasets in the first place. However, the topic of semantic interoperabilty should be further pursued in future projects. The technology to create a proof-of-concept demonstration already exists. The conceptual design of the interoperability functionality is one of the contributions from this project, as well as addressing the third research question: "*How can domain applications effectively utilize a wide range of available open datasets published in different formats?*"

8.2 Conclusion

Open Data is an important tool for governments to communicate with the public, drive innovation and establish new technologies. Current challenges in the scope of open data include the ability to effectively classify, search and reuse the available information. This project addresses these issues, and the contributions have been developed in three modules: i) The development of the Open Transport Data Ontology, which defines the fundamental knowledge organization for use by the other modules, ii) the classification and ontology based search methodologies, which include computational lingustics and iii) the design of an interoperability framework for connecting software applications to datasets and to allow the exhange of information without dealing with heterogenous file formats at the software application level. These modules have been implemented in a "*semantic framework*" that permits different software artifacts to be designed, developed and tested.

The project has demonstrated that by applying an upper level ontology, there is always an unambiguous concept that can relate to any given dataset. The project has further demonstrated that ontology based semantic classification and search give promising results over classical keyword based search, but recommends that the ontology should be extended with a dictionary of domain specific terms. The effectiveness of automatic data classification techniques are sensitive to the quality of the dataset descriptions that are provided by the federated catalogue services and dataset management systems. The semantic interoperability framework for exhanging data and datamodels have been conceptually designed, but not demonstrated in the scope of this project. Semantic interoperability technology is still awaiting a large adaptation, but is a promising field for future studies.

Bibliography

[1] Burwell, S., VanRoekel, S., T., P., & D.J., M. May 2013. Open data policy - managing information as an asset.

[2] Open transport data project website. https://opentransportdata.wordpress.com/. Accessed: 2018-06-03.

[3] Ojo, A., Janowski, T., & Estevez, E. 2009. Semantic interoperability architecture for electronic government. In *Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections Between Citizens, Data and Government*, dg.o '09, 63–72. Digital Government Society of North America. URL: http://dl.acm.org/citation.cfm?id=1556176.1556192.

[4] Vaishnavi, V. & Kuechler, W. 2004. Design Science Research in Information Systems. URL: http://www.desrist.org/desrist.

[5] Rossum, G. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[6] Obe, R. & Hsu, L. 2012. *PostgreSQL: Up and Running*. O'Reilly Media, Inc.

[7] Velasco, R. 2016. *Apache Solr: For Starters*. CreateSpace Independent Publishing Platform, USA.

[8] Robertson, S. & Zaragoza, H. April 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4), 333–389. URL: http://dx.doi.org/10.1561/1500000019, doi:10.1561/1500000019.

[9] Liu, T.-Y. March 2009. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3), 225–331. URL: http://dx.doi.org/10.1561/1500000016, doi:10.1561/1500000016.

[10] Friedman, N., Geiger, D., & Goldszmidt, M. November 1997. Bayesian network classifiers. *Mach. Learn.*, 29(2-3), 131–163. URL: https://doi.org/10.1023/A:1007465528199, doi:10.1023/A:1007465528199.

[11] Farid, D. M., Zhang, L., Rahman, C. M., Hossain, M. A., & Strachan, R. March 2014. Hybrid decision tree and naïve bayes classifiers for multi-class classification tasks. *Expert Syst. Appl.*, 41(4), 1937–1946. URL: http://dx.doi.org/10.1016/j.eswa.2013.08.089, doi:10.1016/j.eswa.2013.08.089.

[12] Takahashi, F. & Abe, S. Nov 2002. Decision-tree-based multiclass support vector machines. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, volume 3, 1418–1422 vol.3. doi:10.1109/ICONIP.2002.1202854.

[13] Cortes, C. & Vapnik, V. September 1995. Support-vector networks. *Mach. Learn.*, 20(3), 273–297. URL: https://doi.org/10.1023/A:1022627411411, doi:10.1023/A:1022627411411.

[14] Altman, N. S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3), 175–185. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879, arXiv:https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879, doi:10.1080/00031305.1992.10475879.

[15] Ou, G. & Murphey, Y. L. 2007. Multi-class pattern classification using neural networks. *Pattern Recognition*, 40(1), 4 – 18. URL: http://www.sciencedirect.com/science/article/pii/S0031320306002081, doi:https://doi.org/10.1016/j.patcog.2006.04.041.

[16] Zhang, M.-L. & Zhou, Z.-H. July 2005. A k-nearest neighbor based algorithm for multi-label classification. In *2005 IEEE International Conference on Granular Computing*, volume 2, 718–721 Vol. 2. doi:10.1109/GRC.2005.1547385.

[17] Vens, C., Struyf, J., Schietgat, L., Džeroski, S., & Blockeel, H. Aug 2008. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2), 185. URL: https://doi.org/10.1007/s10994-008-5077-3, doi:10.1007/s10994-008-5077-3.

[18] Zhang, M.-L. & Zhou, Z.-H. Oct 2006. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10), 1338–1351. doi:10.1109/TKDE.2006.162.

[19] Huang, S.-J., Yu, Y., & Zhou, Z.-H. 2012. Multi-label hypothesis reuse. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 525–533, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2339530.2339615.

[20] Slimani, T. 2013. Description and evaluation of semantic similarity measures approaches. *CoRR*, abs/1310.8059. URL: http://arxiv.org/abs/1310.8059, arXiv:1310.8059.

[21] Rada, R., Mili, H., Bicknell, E., & Blettner, M. Jan 1989. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1), 17–30. doi:10.1109/21.24528.

[22] Wu, Z. & Palmer, M. 1994. Verb semantics and lexical selection. *32nd Annual Meeting of the Association for Computational Linguistics*, 133–138.

[23] WordNet a lexical database for english. https://wordnet.princeton.edu/, note = Accessed: 2018-07-01.

[24] Hagelien, T. F., Chesnokov, A., Johansen, S., Meese, E., & Løvfall, B. 05 2017. Soft: A framework for semantic interoperability of scientific software.

[25] Brickley, D. & Guha, R. RDF schema 1.1. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[26] Schreiber, G. & Gandon, F. RDF 1.1 XML syntax. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[27] Miles, A. & Bechhofer, S. SKOS simple knowledge organization system reference. W3C recommendation, W3C, August 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[28] Dcat application profile for data portals in europe. https://ec.europa.eu/isa2/solutions/dcat-application-profile-data-portals-europe_en. accessed:2018-07-05.

[29] Erickson, J. & Maali, F. Data catalog vocabulary (DCAT). W3C recommendation, W3C, January 2014. http://www.w3.org/TR/2014/REC-vocab-dcat-20140116/.

[30] Miller, G. A. November 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11), 39–41. URL: http://doi.acm.org/10.1145/219717.219748, doi:10.1145/219717.219748.

[31] Arp, R., Smith, B., & Spear, A. 2015. *Building Ontologies with Basic Formal Ontology*. Building Ontologies with Basic Formal Ontology. MIT Press. URL: https://books.google.no/books?id=AUxQCgAAQBAJ.

[32] Daga, A., Cesare, S. d., Lycett, M., & Partridge, C. 2005. An ontological approach for recovering legacy business content. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 08*, HICSS '05, 224.1–, Washington, DC, USA. IEEE Computer Society. URL: http://dx.doi.org/10.1109/HICSS.2005.94, doi:10.1109/HICSS.2005.94.

[33] Cassidy, P. Cosmo ontology. Technical report. accessed: 2018-07-01. URL: https://bartoc.org/en/node/657.

[34] Hearst, M. A. July 1998. Support vector machines. *IEEE Intelligent Systems*, 13(4), 18–28. URL: http://dx.doi.org/10.1109/5254.708428, doi:10.1109/5254.708428.

[35] Dumais, S. T. 2004. *Annual review of information science and technology*, 38, 188–230.

[36] Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. 2009. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition.

[37] Mader, C., Haslhofer, B., & Isaac, A. May 2012. Finding quality issues in skos vocabularies. In *TPDL 2012 Therory and Practice of Digital Libraries*, Germany. URL: http://arxiv.org/pdf/1206.1339v1.

[38] Hagelien, T. F., Jiang, S., & Natvig, M. K. *Manually tagged open transport data Datasets*, 2018. URL: https://github.com/quaat/OTD-semantic-framework/blob/master/doc/manually-tagged-otd-datasets.xlsx?raw=true.

[39] J.W, P., A., K., & M.M, B. 1955. Machine literature searching x. machine language; factors underlying its design and development.

[40] *MUC-4 Evaluation Metrics*, 1992.

A Appendix

skos taxonomy

July 9, 2018

1 The Data Catalogue vocabulary (DCAT)

DCAT is a RDF vocabulary for describing any dataset. A dataset is a collection of data, published or curated by a single source, and available for access or donwnload in one or more formats. DCAT does not include support for semantic relations to concepts, or specification of protocols and parameters for datasets available through APIs

DCAT-AP is an extension of DCAT that combines DCAT with further metadata that allows data to be more easily searchable and discoverable, also supporting multiple languages.

Digi provides a standard for describing datasets and data cataloges (DCAT-AP-NO)[ref] which is based on the DCAT-AP standard [ref].

2 Simple Knowledge Organization System (SKOS)

SKOS is a lightweight vocabulary that formally represents knowledge organization using RDF. DCAT combined with SKOS allows for datasets to be linked to an application taxonomy (application vocabulary).

2.1 Building a open traffic dataset vocabulary (OTD-vocabulary)

Now we have a standardized framework on which we can base the ODT vocabulary. By building the taxonomy on the SKOS vocabulary allows for extending a high-level ontology with domain specific ontologies.

2.2 Automatic classification

The registration of new datasets are often performed by stakeholders with little or no knowledge about the technical side of the information organization and classification schemes in use. Yet it should be reasonable simple for anyone to register new data. In this section the strategies for automatic data classification is discussed and results from studies are presented.

2.3 Keyword based classification

By providing a framework where keywords can be entered together with the datasets, it is possible to match the keywords to the standard vocabulary for concepts. Predefined annotated keywords allows for a direct match, while user-defined keywords can be matched to existing vocabularies by using WordNet as a reference tool, where the semantic similarity can be computed using a thesaurus-based metric such as path similarity, Leacock-Chodorow similarity[ref] or Wu-Palmer Similarity.

2.4 Standard (known) formats classification

Domain specific taxonomies developed to for known dataset structures such as SIRI 2.0 Lite allows for correct classification of relevant datasets.

2.5 Inferred classification based on known schemas

Datasets that contain structured information with embedded meta-data can to some degree be classified based on the interpretation of the meta-data. For XML data with reference to a XML Schema Definition (XSD), the named xml-schema elements can be matched with known vocabularies. Other types of structured data, such as CSV-files often include a column header field that can be matched the same way as the XSD files. JSON/YAML-documents are structured and nested keyword-value pairs. Here the keywords can be treated the same way as the formats indicated above.

Steps:

0. data parsing
1. concept extraction
2. concept matching
3. linking

3 Creating a new instance of the core ODT ontology

The ODT ontology is build on top of the Simple Knowledge Organization System (SKOS) for formal

4 Adding a dataset with a set of keywords

When adding a new dataset, we can make use of the DCAT vocabulary and store it in JSON-LD format.

```
In [17]: from rdflib import Graph, URIRef, BNode, Literal, Namespace
         from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD
         from rdflib.plugins.sparql import prepareQuery

         OTD = Namespace('http://www.quaat.com/ontologies#')
         DCAT = Namespace('http://www.w3.org/ns/dcat#')
         DCT = Namespace('http://purl.org/dc/terms/')
         SKOS = Namespace('http://www.w3.org/2004/02/skos/core#')
         OTDX = Namespace('http://www.quaat.com/ontology/OTDX#')
         QEX = Namespace('http://www.quaat.com/extended_skos#')

         g = Graph()
         g.bind('foaf', FOAF)
```

```
g.bind('dct', DCT)
g.bind('dcat', DCAT)
g.bind('rdf', RDF)
g.bind('dc', DC)
g.bind('skos', SKOS)
g.bind('otd', OTD)
g.bind('qex', QEX)

g.parse('skos.rdf')
#g.parse('dcat-ap_1.1.rdf')
#g.parse('myschema.rdf')
thomas = BNode()
g.add((thomas, RDF.type, FOAF.Person))
g.add((thomas, FOAF.name, Literal("Thomas Hagelien")))

def add_node(name, parent, **kwargs):
    node = URIRef(OTD + name)
    g.add((node, RDF.type, SKOS.Concept))
    g.add((node, SKOS.broader, parent))
    if ('prefLabel' in kwargs):
        for label in kwargs['prefLabel']:
            g.add((node, SKOS.prefLabel, label))
    if ('altLabel' in kwargs):
        for label in kwargs['altLabel']:
            g.add((node, SKOS.altLabel, label))
    g.add((node, SKOS.inScheme, OTDScheme))
    return node

OTDScheme = URIRef(OTD + 'OTDScheme')
g.add((OTDScheme, RDF.type, SKOS.ConceptScheme))
g.add((OTDScheme, DC.title, Literal("Open Transport Datas Taxonomy")))
g.add((OTDScheme, DC.description, Literal("Taxonomy for the classification of Open Tra
g.add((OTDScheme, DC.creator, thomas))

Entity = URIRef(OTD + 'Entity')
g.add((Entity, SKOS.prefLabel, Literal("entity", lang="en")))
g.add((Entity, SKOS.prefLabel, Literal("entitet", lang="nb")))
g.add((Entity, SKOS.topConceptOf, OTDScheme)) # Add a top concept to allow for queryi

Abstraction = add_node('Abstraction',
                       Entity,
                       prefLabel=[Literal("abstraction", lang="en"),
                                  Literal("abstraksjon", lang="nb")])
Communication = add_node('Communication',
                       Abstraction,
                       prefLabel=[Literal("communication", lang="en"),
                                  Literal("kommunikasjon", lang="nb")])
```

```
Agreement = URIRef(OTD + 'Agreement')
g.add((Agreement, RDF.type, SKOS.Concept))
g.add((Agreement, SKOS.broader, Communication))
g.add((Agreement, SKOS.prefLabel, Literal("agreement", lang="en")))
g.add((Agreement, SKOS.prefLabel, Literal("avtale", lang="nb")))
g.add((Agreement, SKOS.inScheme, OTDScheme))

Statistics = URIRef(OTD + 'Statistics')
g.add((Statistics, RDF.type, SKOS.Concept))
g.add((Statistics, SKOS.broader, Abstraction))
g.add((Statistics, SKOS.prefLabel, Literal("statistics", lang="en")))
g.add((Statistics, SKOS.prefLabel, Literal("statistikk", lang="nb")))
g.add((Statistics, SKOS.inScheme, OTDScheme))

APIDescription = URIRef(OTD + 'APIDescription')
g.add((APIDescription, RDF.type, SKOS.Concept))
g.add((APIDescription, SKOS.broader, Abstraction))
g.add((APIDescription, SKOS.prefLabel, Literal("API description", lang="en")))
g.add((APIDescription, SKOS.prefLabel, Literal("grensesnitt", lang="nb")))
g.add((APIDescription, SKOS.inScheme, OTDScheme))

TransportMode = URIRef(OTD + 'TransportMode')
g.add((TransportMode, RDF.type, SKOS.Concept))
g.add((TransportMode, SKOS.broader, Abstraction))
g.add((TransportMode, SKOS.prefLabel, Literal("transport mode", lang="en")))
g.add((TransportMode, SKOS.prefLabel, Literal("transporttype", lang="nb")))
g.add((TransportMode, SKOS.inScheme, OTDScheme))

Air = URIRef(OTD + 'Air')
g.add((Air, RDF.type, SKOS.Concept))
g.add((Air, SKOS.broader, TransportMode))
g.add((Air, SKOS.prefLabel, Literal("air", lang="en")))
g.add((Air, SKOS.prefLabel, Literal("luft", lang="nb")))
g.add((Air, SKOS.inScheme, OTDScheme))

Sea = URIRef(OTD + 'Sea')
g.add((Sea, RDF.type, SKOS.Concept))
g.add((Sea, SKOS.broader, TransportMode))
g.add((Sea, SKOS.prefLabel, Literal("sea", lang="en")))
g.add((Sea, SKOS.prefLabel, Literal("hav", lang="nb")))
g.add((Sea, SKOS.inScheme, OTDScheme))

Rail = URIRef(OTD + 'Rail')
g.add((Rail, RDF.type, SKOS.Concept))
g.add((Rail, SKOS.broader, TransportMode))
g.add((Rail, SKOS.prefLabel, Literal("rail", lang="en")))
g.add((Rail, SKOS.prefLabel, Literal("skinner", lang="nb")))
```

```
g.add((Rail, SKOS.inScheme, OTDScheme))

Cable = URIRef(OTD + 'Cable')
g.add((Cable, RDF.type, SKOS.Concept))
g.add((Cable, SKOS.broader, TransportMode))
g.add((Cable, SKOS.prefLabel, Literal("cable", lang="en")))
g.add((Cable, SKOS.prefLabel, Literal("kabel", lang="nb")))
g.add((Cable, SKOS.inScheme, OTDScheme))

Information = URIRef(OTD + 'Information')
g.add((Information, RDF.type, SKOS.Concept))
g.add((Information, SKOS.broader, Communication))
g.add((Information, SKOS.prefLabel, Literal("information", lang="en")))
g.add((Information, SKOS.prefLabel, Literal("informasjon", lang="nb")))
g.add((Information, SKOS.inScheme, OTDScheme))

GeographicalInformation = URIRef(OTD + 'GeographicalInformation')
g.add((GeographicalInformation, RDF.type, SKOS.Concept))
g.add((GeographicalInformation, SKOS.broader, Communication))
g.add((GeographicalInformation, SKOS.prefLabel, Literal("geographical information", la
g.add((GeographicalInformation, SKOS.prefLabel, Literal("geografisk informasjon", lan
g.add((GeographicalInformation, SKOS.altLabel, Literal("geoinformasjon", lang="nb")))
g.add((GeographicalInformation, SKOS.inScheme, OTDScheme))

Map = URIRef(OTD + 'Map')
g.add((Map, RDF.type, SKOS.Concept))
g.add((Map, SKOS.broader, GeographicalInformation))
g.add((Map, SKOS.prefLabel, Literal("map", lang="en")))
g.add((Map, SKOS.prefLabel, Literal("kart", lang="nb")))
g.add((Map, SKOS.inScheme, OTDScheme))

EnvironmentInformation = URIRef(OTD + 'EnvironmentInformation')
g.add((EnvironmentInformation, RDF.type, SKOS.Concept))
g.add((EnvironmentInformation, SKOS.broader, Information))
g.add((EnvironmentInformation, SKOS.prefLabel, Literal("environment information", lan
g.add((EnvironmentInformation, SKOS.prefLabel, Literal("miljøinformasjuton", lang="nb
g.add((EnvironmentInformation, SKOS.inScheme, OTDScheme))

EmissionInformation = URIRef(OTD + 'EmissionInformation')
g.add((EmissionInformation, RDF.type, SKOS.Concept))
g.add((EmissionInformation, SKOS.broader, EnvironmentInformation))
g.add((EmissionInformation, SKOS.prefLabel, Literal("emission information", lang="en")
g.add((EmissionInformation, SKOS.altLabel, Literal("emission", lang="en")))
g.add((EmissionInformation, SKOS.prefLabel, Literal("utslipp", lang="nb")))
g.add((EmissionInformation, SKOS.altLabel, Literal("utslippsdata", lang="nb")))
g.add((EmissionInformation, SKOS.altLabel, Literal("discharge", lang="nb")))
g.add((EmissionInformation, SKOS.inScheme, OTDScheme))
```

```
MeteorologicalInformation = URIRef(OTD + 'MeteorologicalInformation')
g.add((MeteorologicalInformation, RDF.type, SKOS.Concept))
g.add((MeteorologicalInformation, SKOS.broader, EnvironmentInformation))
g.add((MeteorologicalInformation, SKOS.prefLabel, Literal("meteorological information
#g.add((MeteorologicalInformation, SKOS.altLabel, Literal("weather condition", lang="
g.add((MeteorologicalInformation, SKOS.prefLabel, Literal("meteorologisk informasjon"
g.add((MeteorologicalInformation, SKOS.inScheme, OTDScheme))

HydrometeorologicalInformation = URIRef(OTD + 'HydrometeorologicalInformation')
g.add((HydrometeorologicalInformation, RDF.type, SKOS.Concept))
g.add((HydrometeorologicalInformation, SKOS.broader, EnvironmentInformation))
g.add((HydrometeorologicalInformation, SKOS.prefLabel, Literal("hydrometeorological i
g.add((HydrometeorologicalInformation, SKOS.prefLabel, Literal("hydrometeorologisk in
g.add((HydrometeorologicalInformation, SKOS.inScheme, OTDScheme))

AirQuality = URIRef(OTD + 'AirQuality')
g.add((AirQuality, RDF.type, SKOS.Concept))
g.add((AirQuality, SKOS.broader, EnvironmentInformation))
g.add((AirQuality, SKOS.prefLabel, Literal("air quality", lang="en")))
g.add((AirQuality, SKOS.prefLabel, Literal("luftkvalitet", lang="nb")))
g.add((AirQuality, SKOS.inScheme, OTDScheme))

TravelInformation = URIRef(OTD + 'TravelInformation')
g.add((TravelInformation, RDF.type, SKOS.Concept))
g.add((TravelInformation, SKOS.broader, Information))
g.add((TravelInformation, SKOS.prefLabel, Literal("travel information", lang="en")))
g.add((TravelInformation, SKOS.prefLabel, Literal("reiseinformation", lang="nb")))
g.add((TravelInformation, SKOS.inScheme, OTDScheme))

RealTimeInformation = URIRef(OTD + 'RealTime')
g.add((RealTimeInformation, RDF.type, SKOS.Concept))
g.add((RealTimeInformation, SKOS.broader, TravelInformation))
g.add((RealTimeInformation, SKOS.prefLabel, Literal("real time", lang="en")))
g.add((RealTimeInformation, SKOS.prefLabel, Literal("sanntid", lang="nb")))
g.add((RealTimeInformation, SKOS.inScheme, OTDScheme))

TravelPlan = URIRef(OTD + 'TravelPlan')
g.add((TravelPlan, RDF.type, SKOS.Concept))
g.add((TravelPlan, SKOS.broader, TravelInformation))
g.add((TravelPlan, SKOS.prefLabel, Literal("travel plan", lang="en")))
g.add((TravelPlan, SKOS.prefLabel, Literal("reiseplan", lang="nb")))
g.add((TravelPlan, SKOS.inScheme, OTDScheme))

RoutePlan = URIRef(OTD + 'RoutePlan')
g.add((RoutePlan, RDF.type, SKOS.Concept))
g.add((RoutePlan, SKOS.broader, TravelInformation))
g.add((RoutePlan, SKOS.prefLabel, Literal("route plan", lang="en")))
g.add((RoutePlan, SKOS.prefLabel, Literal("ruteplan", lang="nb")))
```

```
g.add((RoutePlan, SKOS.altLabel, Literal("stoppested", lang="nb")))
g.add((RoutePlan, SKOS.inScheme, OTDScheme))

TrafficInformation = URIRef(OTD + 'TrafficInformation')
g.add((TrafficInformation, RDF.type, SKOS.Concept))
g.add((TrafficInformation, SKOS.broader, Information))
g.add((TrafficInformation, SKOS.prefLabel, Literal("traffic information", lang="en")))
g.add((TrafficInformation, SKOS.prefLabel, Literal("trafikkinformasjon", lang="nb")))
g.add((TrafficInformation, SKOS.inScheme, OTDScheme))

Accident = URIRef(OTD + 'Accident')
g.add((Accident, RDF.type, SKOS.Concept))
g.add((Accident, SKOS.broader, TrafficInformation))
g.add((Accident, SKOS.inScheme, OTDScheme))
g.add((Accident, SKOS.prefLabel, Literal("accident", lang="en")))
g.add((Accident, SKOS.prefLabel, Literal("ulykke", lang="nb")))
g.add((Accident, SKOS.altLabel, Literal("skade", lang="nb")))
g.add((Accident, SKOS.inScheme, OTDScheme))

Incident = URIRef(OTD + 'Incident')
g.add((Incident, RDF.type, SKOS.Concept))
g.add((Incident, SKOS.broader, TrafficInformation))
g.add((Incident, SKOS.prefLabel, Literal("incident", lang="en")))
g.add((Incident, SKOS.prefLabel, Literal("episode", lang="nb")))
g.add((Incident, SKOS.altLabel, Literal("tilfelle", lang="nb")))
g.add((Incident, SKOS.inScheme, OTDScheme))

TrafficCondition = URIRef(OTD + 'TrafficCondition')
g.add((TrafficCondition, RDF.type, SKOS.Concept))
g.add((TrafficCondition, SKOS.broader, TrafficInformation))
g.add((TrafficCondition, SKOS.prefLabel, Literal("traffic condition", lang="en")))
g.add((TrafficCondition, SKOS.prefLabel, Literal("trafikkforhold", lang="nb")))
g.add((TrafficCondition, SKOS.inScheme, OTDScheme))

TransportNetworkCondition = URIRef(OTD + 'TransportNetworkCondition')
g.add((TransportNetworkCondition, RDF.type, SKOS.Concept))
g.add((TransportNetworkCondition, SKOS.broader, TrafficInformation))
g.add((TransportNetworkCondition, SKOS.prefLabel, Literal("transport network conditio
g.add((TransportNetworkCondition, SKOS.prefLabel, Literal("transportnettforhold", lan
g.add((TransportNetworkCondition, SKOS.inScheme, OTDScheme))

TrackingInformation = URIRef(OTD + 'TrackingInformation')
g.add((TrackingInformation, RDF.type, SKOS.Concept))
g.add((TrackingInformation, SKOS.broader, TrafficInformation))
g.add((TrackingInformation, SKOS.prefLabel, Literal("tracking information", lang="en")
g.add((TrackingInformation, SKOS.altLabel, Literal("tracking", lang="en")))
g.add((TrackingInformation, SKOS.prefLabel, Literal("sporing", lang="nb")))
g.add((TrackingInformation, SKOS.inScheme, OTDScheme))
```

```
Prognosis = URIRef(OTD + 'Prognosis')
g.add((Prognosis, RDF.type, SKOS.Concept))
g.add((Prognosis, SKOS.broader, Communication))
g.add((Prognosis, SKOS.prefLabel, Literal("prognosis", lang="en")))
g.add((Prognosis, SKOS.prefLabel, Literal("prognose", lang="nb")))
g.add((Prognosis, SKOS.inScheme, OTDScheme))

Forecast = URIRef(OTD + 'Forecast')
g.add((Forecast, RDF.type, SKOS.Concept))
g.add((Forecast, SKOS.broader, Prognosis))
g.add((Forecast, SKOS.prefLabel, Literal("forecast", lang="en")))
g.add((Forecast, SKOS.prefLabel, Literal("prediksjon", lang="nb")))
g.add((Forecast, SKOS.inScheme, OTDScheme))

WeatherForecast = URIRef(OTD + 'WeatherForecast')
g.add((WeatherForecast, RDF.type, SKOS.Concept))
g.add((WeatherForecast, SKOS.broader, Forecast))
g.add((WeatherForecast, SKOS.prefLabel, Literal("weather forecast", lang="en")))
g.add((WeatherForecast, SKOS.prefLabel, Literal("værmelding", lang="nb")))
g.add((WeatherForecast, SKOS.inScheme, OTDScheme))

Sign = URIRef(OTD + 'Sign')
g.add((Sign, RDF.type, SKOS.Concept))
g.add((Sign, SKOS.broader, Communication))
g.add((Sign, SKOS.prefLabel, Literal("sign", lang="en")))
g.add((Sign, SKOS.prefLabel, Literal("skilt", lang="nb")))
g.add((Sign, SKOS.inScheme, OTDScheme))

StreetSign = URIRef(OTD + 'StreetSign')
g.add((StreetSign, RDF.type, SKOS.Concept))
g.add((StreetSign, SKOS.broader, Sign))
g.add((StreetSign, SKOS.prefLabel, Literal("street sign", lang="en")))
g.add((StreetSign, SKOS.prefLabel, Literal("trafikkskilt", lang="nb")))
g.add((StreetSign, SKOS.inScheme, OTDScheme))

Group = URIRef(OTD + 'Group')
g.add((Group, RDF.type, SKOS.Concept))
g.add((Group, SKOS.broader, Abstraction))
g.add((Group, SKOS.prefLabel, Literal("group", lang="en")))
g.add((Group, SKOS.prefLabel, Literal("gruppe", lang="nb")))
g.add((Group, SKOS.inScheme, OTDScheme))

Collection = URIRef(OTD + 'Collection')
g.add((Collection, RDF.type, SKOS.Concept))
g.add((Collection, SKOS.broader, Group))
g.add((Collection, SKOS.prefLabel, Literal("collection", lang="en")))
```

```
g.add((Collection, SKOS.prefLabel, Literal("samling", lang="nb")))
g.add((Collection, SKOS.inScheme, OTDScheme))

Law = URIRef(OTD + 'Law')
g.add((Law, RDF.type, SKOS.Concept))
g.add((Law, SKOS.broader, Collection))
g.add((Law, SKOS.prefLabel, Literal("law", lang="en")))
g.add((Law, SKOS.prefLabel, Literal("lov", lang="nb")))
g.add((Law, SKOS.inScheme, OTDScheme))

Traffic = URIRef(OTD + 'Traffic')
g.add((Traffic, RDF.type, SKOS.Concept))
g.add((Traffic, SKOS.broader, Collection))
g.add((Traffic, SKOS.prefLabel, Literal("traffic", lang="en")))
g.add((Traffic, SKOS.prefLabel, Literal("trafikk", lang="nb")))
g.add((Traffic, SKOS.inScheme, OTDScheme))

Formation = URIRef(OTD + 'Formation')
g.add((Formation, RDF.type, SKOS.Concept))
g.add((Formation, SKOS.broader, Group))
g.add((Formation, SKOS.prefLabel, Literal("formation", lang="en")))
g.add((Formation, SKOS.prefLabel, Literal("formasjon", lang="nb")))
g.add((Formation, SKOS.inScheme, OTDScheme))

TrafficFlow = URIRef(OTD + 'TrafficFlow')
g.add((TrafficFlow, RDF.type, SKOS.Concept))
g.add((TrafficFlow, SKOS.broader, Formation))
g.add((TrafficFlow, SKOS.prefLabel, Literal("traffic flow", lang="en")))
g.add((TrafficFlow, SKOS.prefLabel, Literal("trafikkflyt", lang="nb")))
g.add((TrafficFlow, SKOS.inScheme, OTDScheme))

TrafficQueue = URIRef(OTD + 'TrafficQueue')
g.add((TrafficQueue, RDF.type, SKOS.Concept))
g.add((TrafficQueue, SKOS.broader, Formation))
g.add((TrafficQueue, SKOS.prefLabel, Literal("traffic queue", lang="en")))
g.add((TrafficQueue, SKOS.prefLabel, Literal("kø", lang="nb")))
g.add((TrafficQueue, SKOS.inScheme, OTDScheme))

Organization = URIRef(OTD + 'Organization')
g.add((Organization, RDF.type, SKOS.Concept))
g.add((Organization, SKOS.broader, Group))
g.add((Organization, SKOS.prefLabel, Literal("organization", lang="en")))
g.add((Organization, SKOS.prefLabel, Literal("organisasjon", lang="nb")))
g.add((Organization, SKOS.inScheme, OTDScheme))

Company = URIRef(OTD + 'Company')
g.add((Company, RDF.type, SKOS.Concept))
g.add((Company, SKOS.broader, Organization))
```

```
g.add((Company, SKOS.prefLabel, Literal("company", lang="en")))
g.add((Company, SKOS.prefLabel, Literal("selskap", lang="nb")))
g.add((Company, SKOS.inScheme, OTDScheme))

Object = URIRef(OTD + 'Object')
g.add((Object, RDF.type, SKOS.Concept))
g.add((Object, SKOS.broader, OTD.Entity))
g.add((Object, SKOS.prefLabel, Literal("object", lang="en")))
g.add((Object, SKOS.prefLabel, Literal("objekt", lang="nb")))
g.add((Object, SKOS.inScheme, OTDScheme))

Artifact = URIRef(OTD + 'Artifact')
g.add((Artifact, RDF.type, SKOS.Concept))
g.add((Artifact, SKOS.broader, Object))
g.add((Artifact, SKOS.prefLabel, Literal("artifact", lang="en")))
g.add((Artifact, SKOS.prefLabel, Literal("gjenstand", lang="nb")))
g.add((Artifact, SKOS.inScheme, OTDScheme))

Facility = URIRef(OTD + 'Facility')
g.add((Facility, RDF.type, SKOS.Concept))
g.add((Facility, SKOS.broader, Artifact))
g.add((Facility, SKOS.prefLabel, Literal("facility", lang="en")))
g.add((Facility, SKOS.prefLabel, Literal("anlegg", lang="nb")))
g.add((Facility, SKOS.inScheme, OTDScheme))

Airport = URIRef(OTD + 'Airport')
g.add((Airport, RDF.type, SKOS.Concept))
g.add((Airport, SKOS.broader, Facility))
g.add((Airport, SKOS.prefLabel, Literal("airport", lang="en")))
g.add((Airport, SKOS.prefLabel, Literal("flyplass", lang="nb")))
g.add((Airport, SKOS.inScheme, OTDScheme))

BusTerminal = URIRef(OTD + 'BusTerminal')
g.add((BusTerminal, RDF.type, SKOS.Concept))
g.add((BusTerminal, SKOS.broader, Terminal))
g.add((BusTerminal, SKOS.prefLabel, Literal("bus terminal", lang="en")))
g.add((BusTerminal, SKOS.altLabel, Literal("bus station", lang="en")))
g.add((BusTerminal, SKOS.altLabel, Literal("busstasjon", lang="nb")))
g.add((BusTerminal, SKOS.inScheme, OTDScheme))


Representation = URIRef(OTD + 'Representation')
g.add((Representation, RDF.type, SKOS.Concept))
g.add((Representation, SKOS.broader, Artifact))
g.add((Representation, SKOS.prefLabel, Literal("representation", lang="en")))
g.add((Representation, SKOS.prefLabel, Literal("representasjon", lang="nb")))
g.add((Representation, SKOS.inScheme, OTDScheme))
```

```
Structure = URIRef(OTD + 'Structure')
g.add((Structure, RDF.type, SKOS.Concept))
g.add((Structure, SKOS.broader, Artifact))
g.add((Structure, SKOS.prefLabel, Literal("structure", lang="en")))
g.add((Structure, SKOS.prefLabel, Literal("struktur", lang="nb")))
g.add((Structure, SKOS.inScheme, OTDScheme))

TransportNetwork = URIRef(OTD + 'TransportNetwork')
g.add((TransportNetwork, RDF.type, SKOS.Concept))
g.add((TransportNetwork, SKOS.broader, Structure))
g.add((TransportNetwork, SKOS.prefLabel, Literal("transport network", lang="en")))
g.add((TransportNetwork, SKOS.prefLabel, Literal("transportnettverk", lang="nb")))
g.add((TransportNetwork, SKOS.inScheme, OTDScheme))

Bridge = URIRef(OTD + 'Bridge')
g.add((Bridge, RDF.type, SKOS.Concept))
g.add((Bridge, SKOS.broader, Structure))
g.add((Bridge, SKOS.prefLabel, Literal("bridge", lang="en")))
g.add((Bridge, SKOS.prefLabel, Literal("bro", lang="nb")))
g.add((Bridge, SKOS.inScheme, OTDScheme))

Building = URIRef(OTD + 'Building')
g.add((Building, RDF.type, SKOS.Concept))
g.add((Building, SKOS.broader, Structure))
g.add((Building, SKOS.prefLabel, Literal("building", lang="en")))
g.add((Building, SKOS.prefLabel, Literal("bygning", lang="nb")))
g.add((Building, SKOS.inScheme, OTDScheme))

Garage = URIRef(OTD + 'Garage')
g.add((Garage, RDF.type, SKOS.Concept))
g.add((Garage, SKOS.broader, Building))
g.add((Garage, SKOS.prefLabel, Literal("garage", lang="en")))
g.add((Garage, SKOS.prefLabel, Literal("garasje", lang="nb")))
g.add((Garage, SKOS.inScheme, OTDScheme))

Booth = URIRef(OTD + 'Booth')
g.add((Booth, RDF.type, SKOS.Concept))
g.add((Booth, SKOS.broader, Structure))
g.add((Booth, SKOS.prefLabel, Literal("booth", lang="en")))
g.add((Booth, SKOS.prefLabel, Literal("skur", lang="nb")))
g.add((Booth, SKOS.inScheme, OTDScheme))

Toolbooth = URIRef(OTD + 'Toolbooth')
g.add((Toolbooth, RDF.type, SKOS.Concept))
g.add((Toolbooth, SKOS.broader, Booth))
g.add((Toolbooth, SKOS.prefLabel, Literal("tool booth", lang="en")))
g.add((Toolbooth, SKOS.prefLabel, Literal("bomstasjon", lang="nb")))
g.add((Toolbooth, SKOS.inScheme, OTDScheme))
```

```
Vehicle = URIRef(OTD + 'Vehicle')
g.add((Vehicle, RDF.type, SKOS.Concept))
g.add((Vehicle, SKOS.broader, Artifact))
g.add((Vehicle, SKOS.prefLabel, Literal("vehicle", lang="en")))
g.add((Vehicle, SKOS.prefLabel, Literal("kjøretøy", lang="nb")))
g.add((Vehicle, SKOS.inScheme, OTDScheme))

Way = URIRef(OTD + 'Way')
g.add((Way, RDF.type, SKOS.Concept))
g.add((Way, SKOS.broader, Artifact))
g.add((Way, SKOS.prefLabel, Literal("way", lang="en")))
g.add((Way, SKOS.prefLabel, Literal("strekning", lang="nb")))
g.add((Way, SKOS.inScheme, OTDScheme))

Road = URIRef(OTD + 'Road')
g.add((Road, RDF.type, SKOS.Concept))
g.add((Road, SKOS.broader, Way))
g.add((Road, SKOS.prefLabel, Literal("road", lang="en")))
g.add((Road, SKOS.prefLabel, Literal("vei", lang="nb")))
g.add((Road, SKOS.inScheme, OTDScheme))

Trail = URIRef(OTD + 'Trail')
g.add((Trail, RDF.type, SKOS.Concept))
g.add((Trail, SKOS.broader, Way))
g.add((Trail, SKOS.prefLabel, Literal("trail", lang="en")))
g.add((Trail, SKOS.prefLabel, Literal("sti", lang="nb")))
g.add((Trail, SKOS.inScheme, OTDScheme))

Detour = URIRef(OTD + 'Detour')
g.add((Detour, RDF.type, SKOS.Concept))
g.add((Detour, SKOS.broader, Road))
g.add((Detour, SKOS.prefLabel, Literal("detour", lang="en")))
g.add((Detour, SKOS.prefLabel, Literal("omvei", lang="nb")))
g.add((Detour, SKOS.altLabel, Literal("avstikker", lang="nb")))
g.add((Detour, SKOS.inScheme, OTDScheme))

Highway = URIRef(OTD + 'Highway')
g.add((Highway, RDF.type, SKOS.Concept))
g.add((Highway, SKOS.broader, Road))
g.add((Highway, SKOS.prefLabel, Literal("highway", lang="en")))
g.add((Highway, SKOS.prefLabel, Literal("hovedvei", lang="nb")))
g.add((Highway, SKOS.altLabel, Literal("motorvei", lang="nb")))
g.add((Highway, SKOS.inScheme, OTDScheme))

Roadway = URIRef(OTD + 'Roadway')
g.add((Roadway, RDF.type, SKOS.Concept))
g.add((Roadway, SKOS.broader, Road))
```

```
g.add((Roadway, SKOS.prefLabel, Literal("roadway", lang="en")))
g.add((Roadway, SKOS.prefLabel, Literal("landevei", lang="nb")))
g.add((Roadway, SKOS.inScheme, OTDScheme))

Speedway = URIRef(OTD + 'Speedway')
g.add((Speedway, RDF.type, SKOS.Concept))
g.add((Speedway, SKOS.broader, Road))
g.add((Speedway, SKOS.prefLabel, Literal("speedway", lang="en")))
g.add((Speedway, SKOS.prefLabel, Literal("speedway", lang="nb")))
g.add((Speedway, SKOS.inScheme, OTDScheme))

Tunnel = URIRef(OTD + 'Tunnel')
g.add((Tunnel, RDF.type, SKOS.Concept))
g.add((Tunnel, SKOS.broader, Way))
g.add((Tunnel, SKOS.prefLabel, Literal("tunnel", lang="en")))
g.add((Tunnel, SKOS.prefLabel, Literal("tunnel", lang="nb")))
g.add((Tunnel, SKOS.inScheme, OTDScheme))

Location = URIRef(OTD + 'Location')
g.add((Location, RDF.type, SKOS.Concept))
g.add((Location, SKOS.broader, Object))
g.add((Location, SKOS.prefLabel, Literal("location", lang="en")))
g.add((Location, SKOS.prefLabel, Literal("sted", lang="nb")))
g.add((Location, SKOS.altLabel, Literal("lokasjon", lang="nb")))
g.add((Location, SKOS.altLabel, Literal("beliggenhet", lang="nb")))
g.add((Location, SKOS.inScheme, OTDScheme))

POI = URIRef(OTD + 'POI')
g.add((POI, RDF.type, SKOS.Concept))
g.add((POI, SKOS.broader, Object))
g.add((POI, SKOS.inScheme, OTDScheme))
g.add((POI, SKOS.prefLabel, Literal("point of interest", lang="en")))
g.add((POI, SKOS.altLabel, Literal("poi", lang="en")))
g.add((POI, SKOS.prefLabel, Literal("interessepunkt", lang="nb")))

Region = URIRef(OTD + 'Region')
g.add((Region, RDF.type, SKOS.Concept))
g.add((Region, SKOS.broader, Location))
g.add((Region, SKOS.prefLabel, Literal("region", lang="en")))
g.add((Region, SKOS.prefLabel, Literal("region", lang="nb")))
g.add((Region, SKOS.inScheme, OTDScheme))

TopographicPoint = URIRef(OTD + 'TopographicPoint')
g.add((TopographicPoint, RDF.type, SKOS.Concept))
g.add((TopographicPoint, SKOS.broader, Location))
g.add((TopographicPoint, SKOS.prefLabel, Literal("topographic point", lang="en")))
g.add((TopographicPoint, SKOS.prefLabel, Literal("topografisk punkt", lang="nb")))
g.add((TopographicPoint, SKOS.inScheme, OTDScheme))
```

```
Junction = URIRef(OTD + 'Junction')
g.add((Junction, RDF.type, SKOS.Concept))
g.add((Junction, SKOS.broader, TopographicPoint))
g.add((Junction, SKOS.prefLabel, Literal("junction", lang="en")))
g.add((Junction, SKOS.prefLabel, Literal("veikryss", lang="nb")))
g.add((Junction, SKOS.inScheme, OTDScheme))

Interchange = URIRef(OTD + 'Interchange')
g.add((Interchange, RDF.type, SKOS.Concept))
g.add((Interchange, SKOS.broader, Junction))
g.add((Interchange, SKOS.prefLabel, Literal("interchange", lang="en")))
g.add((Interchange, SKOS.prefLabel, Literal("utveksling", lang="nb")))
g.add((Interchange, SKOS.inScheme, OTDScheme))

Intersection = URIRef(OTD + 'Intersection')
g.add((Intersection, RDF.type, SKOS.Concept))
g.add((Intersection, SKOS.broader, Junction))
g.add((Intersection, SKOS.prefLabel, Literal("intersection", lang="en")))
g.add((Intersection, SKOS.prefLabel, Literal("kryss", lang="nb")))
g.add((Intersection, SKOS.inScheme, OTDScheme))

RailwayJunction = URIRef(OTD + 'RailwayJunction')
g.add((RailwayJunction, RDF.type, SKOS.Concept))
g.add((RailwayJunction, SKOS.broader, Junction))
g.add((RailwayJunction, SKOS.prefLabel, Literal("railway junction", lang="en")))
g.add((RailwayJunction, SKOS.prefLabel, Literal("jernbaneovergang", lang="nb")))
g.add((RailwayJunction, SKOS.inScheme, OTDScheme))

TJunction = URIRef(OTD + 'TJunction')
g.add((TJunction, RDF.type, SKOS.Concept))
g.add((TJunction, SKOS.broader, Junction))
g.add((TJunction, SKOS.prefLabel, Literal("t junction", lang="en")))
g.add((TJunction, SKOS.altLabel, Literal("3-way junction", lang="en")))
g.add((TJunction, SKOS.prefLabel, Literal("3-veis kryss", lang="nb")))
g.add((TJunction, SKOS.inScheme, OTDScheme))

TrafficCircle = URIRef(OTD + 'TrafficCircle')
g.add((TrafficCircle, RDF.type, SKOS.Concept))
g.add((TrafficCircle, SKOS.broader, Junction))
g.add((TrafficCircle, SKOS.prefLabel, Literal("traffic circle", lang="en")))
g.add((TrafficCircle, SKOS.prefLabel, Literal("rundkjøring", lang="nb")))
g.add((TrafficCircle, SKOS.inScheme, OTDScheme))

ServiceArea = URIRef(OTD + 'ServiceArea')
g.add((ServiceArea, RDF.type, SKOS.Concept))
g.add((ServiceArea, SKOS.broader, TopographicPoint))
g.add((ServiceArea, SKOS.prefLabel, Literal("service area", lang="en")))
```

```
g.add((ServiceArea, SKOS.prefLabel, Literal("serviceområde", lang="nb")))
g.add((ServiceArea, SKOS.inScheme, OTDScheme))

TransferNode = URIRef(OTD + 'TransferNode')
g.add((TransferNode, RDF.type, SKOS.Concept))
g.add((TransferNode, SKOS.broader, TopographicPoint))
g.add((TransferNode, SKOS.prefLabel, Literal("transfer node", lang="en")))
g.add((TransferNode, SKOS.altLabel, Literal("transfer point", lang="en")))
g.add((TransferNode, SKOS.prefLabel, Literal("overføringspunkt", lang="nb")))
g.add((TransferNode, SKOS.inScheme, OTDScheme))

Terminal = URIRef(OTD + 'Terminal')
g.add((Terminal, RDF.type, SKOS.Concept))
g.add((Terminal, SKOS.broader, TransferNode))
g.add((Terminal, SKOS.prefLabel, Literal("terminal", lang="en")))
g.add((Terminal, SKOS.prefLabel, Literal("terminal", lang="nb")))
g.add((Terminal, SKOS.inScheme, OTDScheme))


Parking = URIRef(OTD + 'Parking')
g.add((Parking, RDF.type, SKOS.Concept))
g.add((Parking, SKOS.broader, TransferNode))
g.add((Parking, SKOS.prefLabel, Literal("parking", lang="en")))
g.add((Parking, SKOS.prefLabel, Literal("parkering", lang="nb")))
g.add((Parking, SKOS.inScheme, OTDScheme))

Park_and_Ride = URIRef(OTD + 'Park_and_Ride')
g.add((Park_and_Ride, RDF.type, SKOS.Concept))
g.add((Park_and_Ride, SKOS.broader, TransferNode))
g.add((Park_and_Ride, SKOS.prefLabel, Literal("park and ride", lang="en")))
g.add((Park_and_Ride, SKOS.altLabel, Literal("coaching inn", lang="en")))
g.add((Park_and_Ride, SKOS.prefLabel, Literal("skysstasjon", lang="nb")))
g.add((Park_and_Ride, SKOS.inScheme, OTDScheme))

Zone = URIRef(OTD + 'Zone')
g.add((Zone, RDF.type, SKOS.Concept))
g.add((Zone, SKOS.broader, TopographicPoint))
g.add((Zone, SKOS.prefLabel, Literal("zone", lang="en")))
g.add((Zone, SKOS.prefLabel, Literal("sone", lang="nb")))
g.add((Zone, SKOS.inScheme, OTDScheme))

Organism = URIRef(OTD + 'Organism')
g.add((Organism, RDF.type, SKOS.Concept))
g.add((Organism, SKOS.broader, Object))
g.add((Organism, SKOS.prefLabel, Literal("organism", lang="en")))
g.add((Organism, SKOS.prefLabel, Literal("organisme", lang="nb")))
g.add((Organism, SKOS.inScheme, OTDScheme))
```

```
Process = URIRef(OTD + 'Process')
g.add((Process, RDF.type, SKOS.Concept))
g.add((Process, SKOS.broader, Entity))
g.add((Process, SKOS.prefLabel, Literal("process", lang="en")))
g.add((Process, SKOS.prefLabel, Literal("prosess", lang="nb")))
g.add((Process, SKOS.inScheme, OTDScheme))

TransportService = URIRef(OTD + 'TransportService')
g.add((TransportService, RDF.type, SKOS.Concept))
g.add((TransportService, SKOS.broader, Process))
g.add((TransportService, SKOS.prefLabel, Literal("transport service", lang="en")))
g.add((TransportService, SKOS.prefLabel, Literal("transporttjeneste", lang="nb")))
g.add((TransportService, SKOS.inScheme, OTDScheme))

CityBike = URIRef(OTD + 'CityBike')
g.add((CityBike, RDF.type, SKOS.Concept))
g.add((CityBike, SKOS.broader, OTD.TransportService))
g.add((CityBike, SKOS.prefLabel, Literal("city bike", lang="en")))
g.add((CityBike, SKOS.prefLabel, Literal("bysykkel", lang="nb")))
g.add((CityBike, SKOS.inScheme, OTD.OTDScheme))

Taxi = URIRef(OTD + 'Taxi')
g.add((Taxi, RDF.type, SKOS.Concept))
g.add((Taxi, SKOS.broader, OTD.TransportService))
g.add((Taxi, SKOS.prefLabel, Literal("taxi", lang="en")))
g.add((Taxi, SKOS.altLabel, Literal("drosje", lang="en")))
g.add((Taxi, SKOS.prefLabel, Literal("taxi", lang="nb")))
g.add((Taxi, SKOS.inScheme, OTD.OTDScheme))

CarSharing = URIRef(OTD + 'CarSharing')
g.add((CarSharing, RDF.type, SKOS.Concept))
g.add((CarSharing, SKOS.broader, TransportService))
g.add((CarSharing, SKOS.prefLabel, Literal("carsharing", lang="en")))
g.add((CarSharing, SKOS.altLabel, Literal("car pooling", lang="en")))
g.add((CarSharing, SKOS.prefLabel, Literal("bildeling", lang="nb")))
g.add((CarSharing, SKOS.inScheme, OTDScheme))

Event = URIRef(OTD + 'Event')
g.add((Event, RDF.type, SKOS.Concept))
g.add((Event, SKOS.broader, Process))
g.add((Event, SKOS.prefLabel, Literal("event", lang="en")))
g.add((Event, SKOS.prefLabel, Literal("begivenhet", lang="nb")))
g.add((Event, SKOS.inScheme, OTDScheme))

Action = URIRef(OTD + 'Action')
g.add((Action, RDF.type, SKOS.Concept))
g.add((Action, SKOS.broader, Event))
g.add((Action, SKOS.prefLabel, Literal("action", lang="en")))
```

```
g.add((Action, SKOS.prefLabel, Literal("handling", lang="nb")))
g.add((Action, SKOS.inScheme, OTDScheme))

Activity = URIRef(OTD + 'Activity')
g.add((Activity, RDF.type, SKOS.Concept))
g.add((Activity, SKOS.broader, Event))
g.add((Activity, SKOS.prefLabel, Literal("activity", lang="en")))
g.add((Activity, SKOS.prefLabel, Literal("aktivitet", lang="nb")))
g.add((Activity, SKOS.inScheme, OTDScheme))

Happening = URIRef(OTD + 'Happening')
g.add((Happening, RDF.type, SKOS.Concept))
g.add((Happening, SKOS.broader, Event))
g.add((Happening, SKOS.prefLabel, Literal("happening", lang="en")))
g.add((Happening, SKOS.prefLabel, Literal("hendelse", lang="nb")))
g.add((Happening, SKOS.inScheme, OTDScheme))

#Accident = URIRef(OTD + 'Accident')
#g.add((Accident, RDF.type, SKOS.Concept))
#g.add((Accident, SKOS.broader, Happening))
#g.add((Accident, SKOS.prefLabel, Literal("accident", lang="en")))
#g.add((Accident, SKOS.prefLabel, Literal("ulykke", lang="nb")))
#g.add((Accident, SKOS.inScheme, OTDScheme))

#Incident = URIRef(OTD + 'Incident')
#g.add((Incident, RDF.type, SKOS.Concept))
#g.add((Incident, SKOS.broader, Happening))
#g.add((Incident, SKOS.prefLabel, Literal("incident", lang="en")))
#g.add((Incident, SKOS.prefLabel, Literal("episode", lang="nb")))
#g.add((Incident, SKOS.inScheme, OTDScheme))

#TrafficCondition = URIRef(OTD + 'TrafficCondition')
#g.add((TrafficCondition, RDF.type, SKOS.Concept))
#g.add((TrafficCondition, SKOS.broader, Happening))
#g.add((TrafficCondition, SKOS.prefLabel, Literal("traffic condition", lang="en")))
#g.add((TrafficCondition, SKOS.prefLabel, Literal("trafikkforhold", lang="nb")))
#g.add((TrafficCondition, SKOS.inScheme, OTDScheme))

Phenomenon = URIRef(OTD + 'Phenomenon')
g.add((Phenomenon, RDF.type, SKOS.Concept))
g.add((Phenomenon, SKOS.broader, Process))
g.add((Phenomenon, SKOS.prefLabel, Literal("phenomenon", lang="en")))
g.add((Phenomenon, SKOS.prefLabel, Literal("fenomen", lang="nb")))
g.add((Phenomenon, SKOS.inScheme, OTDScheme))

#TransportNetworkCondition = URIRef(OTD + 'TransportNetworkCondition')
#g.add((TransportNetworkCondition, RDF.type, SKOS.Concept))
#g.add((TransportNetworkCondition, SKOS.broader, Phenomenon))
```

```
#g.add((TransportNetworkCondition, SKOS.prefLabel, Literal("transport network conditi
#g.add((TransportNetworkCondition, SKOS.prefLabel, Literal("transportnettilstand", la
#g.add((TransportNetworkCondition, SKOS.inScheme, OTDScheme))

WeatherCondition = URIRef(OTD + 'WeatherCondition')
g.add((WeatherCondition, RDF.type, SKOS.Concept))
g.add((WeatherCondition, SKOS.broader, Phenomenon))
g.add((WeatherCondition, SKOS.prefLabel, Literal("weather condition", lang="en")))
g.add((WeatherCondition, SKOS.prefLabel, Literal("værforhold", lang="nb")))
g.add((WeatherCondition, SKOS.inScheme, OTDScheme))

Release = URIRef(OTD + 'Release')
g.add((Release, RDF.type, SKOS.Concept))
g.add((Release, SKOS.broader, Process))
g.add((Release, SKOS.prefLabel, Literal("release", lang="en")))
g.add((Release, SKOS.prefLabel, Literal("frigjøre", lang="nb")))
g.add((Release, SKOS.definition, Literal("the production and discharge of something, 
g.add((Release, SKOS.inScheme, OTDScheme))

Line = URIRef(OTD + 'Line')
g.add((Line, RDF.type, SKOS.Concept))
g.add((Line, SKOS.broader, OTD.Location))
g.add((Line, SKOS.prefLabel, Literal("line", lang="en")))
g.add((Line, SKOS.prefLabel, Literal("linje", lang="nb")))
g.add((Line, SKOS.inScheme, OTD.OTDScheme))

Route = URIRef(OTD + 'Route')
g.add((Route, RDF.type, SKOS.Concept))
g.add((Route, SKOS.broader, OTD.Line))
g.add((Route, SKOS.prefLabel, Literal("route", lang="en")))
g.add((Route, SKOS.prefLabel, Literal("rute", lang="nb")))
g.add((Route, SKOS.altLabel, Literal("trasse", lang="nb")))
g.add((Route, SKOS.inScheme, OTD.OTDScheme))

Schedule = URIRef(OTD + 'Schedule')
g.add((Schedule, RDF.type, SKOS.Concept))
g.add((Schedule, SKOS.broader, OTD.TravelInformation))
g.add((Schedule, SKOS.prefLabel, Literal("schedule", lang="en")))
g.add((Schedule, SKOS.prefLabel, Literal("agenda", lang="nb")))
g.add((Schedule, SKOS.inScheme, OTD.OTDScheme))

Timetable = URIRef(OTD + 'Timetable')
g.add((Timetable, RDF.type, SKOS.Concept))
g.add((Timetable, SKOS.broader, OTD.Schedule))
g.add((Timetable, SKOS.prefLabel, Literal("timetable", lang="en")))
g.add((Timetable, SKOS.prefLabel, Literal("rutetabell", lang="nb")))
g.add((Timetable, SKOS.inScheme, OTD.OTDScheme))
```

```
PublicTransport = URIRef(OTD + 'PublicTransport')
g.add((PublicTransport, RDF.type, SKOS.Concept))
g.add((PublicTransport, SKOS.broader, OTD.Vehicle))
g.add((PublicTransport, SKOS.prefLabel, Literal("public transport", lang="en")))
g.add((PublicTransport, SKOS.prefLabel, Literal("offentlig transport", lang="nb")))
g.add((PublicTransport, SKOS.altLabel, Literal("transport", lang="nb")))
g.add((PublicTransport, SKOS.inScheme, OTD.OTDScheme))

Train = URIRef(OTD + 'Train')
g.add((Train, RDF.type, SKOS.Concept))
g.add((Train, SKOS.broader, OTD.PublicTransport))
g.add((Train, SKOS.prefLabel, Literal("train", lang="en")))
g.add((Train, SKOS.prefLabel, Literal("tog", lang="nb")))
g.add((Train, SKOS.altLabel, Literal("jernbane", lang="nb")))
g.add((Train, SKOS.inScheme, OTD.OTDScheme))

Bus = URIRef(OTD + 'Bus')
g.add((Bus, RDF.type, SKOS.Concept))
g.add((Bus, SKOS.broader, OTD.PublicTransport))
g.add((Bus, SKOS.prefLabel, Literal("bus", lang="en")))
g.add((Bus, SKOS.prefLabel, Literal("buss", lang="nb")))
g.add((Bus, SKOS.inScheme, OTD.OTDScheme))

Boat = URIRef(OTD + 'Boat')
g.add((Boat, RDF.type, SKOS.Concept))
g.add((Boat, SKOS.broader, OTD.PublicTransport))
g.add((Boat, SKOS.prefLabel, Literal("boat", lang="en")))
g.add((Boat, SKOS.prefLabel, Literal(u"båt", lang="nb")))
g.add((Boat, SKOS.inScheme, OTD.OTDScheme))

PersonalTransport = URIRef(OTD + 'PersonalTransport')
g.add((PersonalTransport, RDF.type, SKOS.Concept))
g.add((PersonalTransport, SKOS.broader, OTD.Vehicle))
g.add((PersonalTransport, SKOS.prefLabel, Literal("personal transport", lang="en")))
g.add((PersonalTransport, SKOS.prefLabel, Literal("persolig transport", lang="nb")))
g.add((PersonalTransport, SKOS.altLabel, Literal("skyss", lang="nb")))
g.add((PersonalTransport, SKOS.inScheme, OTD.OTDScheme))

Bicycle = URIRef(OTD + 'Bicycle')
g.add((Bicycle, RDF.type, SKOS.Concept))
g.add((Bicycle, SKOS.broader, OTD.PersonalTransport))
g.add((Bicycle, SKOS.prefLabel, Literal("bicycle", lang="en")))
g.add((Bicycle, SKOS.altLabel, Literal("bike", lang="en")))
g.add((Bicycle, SKOS.prefLabel, Literal("sykkel", lang="nb")))
g.add((Bicycle, SKOS.inScheme, OTD.OTDScheme))

Plan = URIRef(OTD + 'Plan')
g.add((Plan, RDF.type, SKOS.Concept))
```

```python
g.add((Plan, SKOS.broader, OTD.Representation))
g.add((Plan, SKOS.prefLabel, Literal("plan", lang="en")))
g.add((Plan, SKOS.prefLabel, Literal(u"plan", lang="nb")))
g.add((Plan, SKOS.inScheme, OTD.OTDScheme))

RoutePlan = URIRef(OTD + 'RoutePlan')
g.add((RoutePlan, RDF.type, SKOS.Concept))
g.add((RoutePlan, SKOS.broader, OTD.Plan))
g.add((RoutePlan, SKOS.prefLabel, Literal("route plan", lang="en")))
g.add((RoutePlan, SKOS.prefLabel, Literal(u"ruteplan", lang="nb")))
g.add((RoutePlan, SKOS.inScheme, OTD.OTDScheme))

BusStop = URIRef(OTD + 'BusStop')
g.add((BusStop, RDF.type, SKOS.Concept))
g.add((BusStop, SKOS.broader, OTD.TransferNode))
g.add((BusStop, SKOS.prefLabel, Literal("bus stop", lang="en")))
g.add((BusStop, SKOS.prefLabel, Literal(u"busstopp", lang="nb")))
g.add((BusStop, SKOS.inScheme, OTD.OTDScheme))

Station = URIRef(OTD + 'Station')
g.add((Station, RDF.type, SKOS.Concept))
g.add((Station, SKOS.broader, OTD.TransferNode))
g.add((Station, SKOS.prefLabel, Literal("station", lang="en")))
g.add((Station, SKOS.prefLabel, Literal(u"stasjon", lang="nb")))
g.add((Station, SKOS.inScheme, OTD.OTDScheme))

ChargingStation = URIRef(OTD + 'ChargingStation')
g.add((ChargingStation, RDF.type, SKOS.Concept))
g.add((ChargingStation, SKOS.broader, OTD.Station))
g.add((ChargingStation, SKOS.prefLabel, Literal("charging station", lang="en")))
g.add((ChargingStation, SKOS.altLabel, Literal("charging", lang="en")))
#g.add((ChargingStation, SKOS.altLabel, Literal("station", lang="en")))
g.add((ChargingStation, SKOS.altLabel, Literal("charging service", lang="en")))
g.add((ChargingStation, SKOS.altLabel, Literal("filling station", lang="en")))
g.add((ChargingStation, SKOS.prefLabel, Literal(u"ladestasjon", lang="nb")))
g.add((ChargingStation, SKOS.altLabel, Literal(u"lading", lang="nb")))
g.add((ChargingStation, SKOS.inScheme, OTD.OTDScheme))

g.serialize(destination="otd-ontology-v6.owl", format="xml")
```

B Appendix

Automatic classification, compute similarity scores

July 8, 2018

```
In [1]: import requests
        from rdflib import Graph
        from rdflib import Namespace
        from rdflib import URIRef
        from rdflib import Literal
        from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD, SKOS
        from bs4 import BeautifulSoup

        ODT = Namespace('http://www.quaat.com/ontologies#')
        DCAT = Namespace('http://www.w3.org/ns/dcat#')
        DCT = Namespace('http://purl.org/dc/terms/')
        ODTX = Namespace('http://www.quaat.com/ontology/ODTX#')
        QEX = Namespace('http://www.quaat.com/extended_skos#')

        # Get a list of all datasets
        r = requests.get('http://78.91.98.234:5000/api/3/action/package_list')
        json = r.json()

        def get_value(dict, key):
            if key in dict:
                return dict[key]
            return ""

        corpus = [] # the collection of texts
        dataset_ref = []  # a list of datasets corresponding to the corpus

        # For each item in the list of dataset, extract the contents
        for dataset in json['result']:
            r2 = requests.get('http://78.91.98.234:5000/dataset/' + dataset + '.rdf')
            graph = Graph()
            graph.parse(format='xml', data=r2.text)

            # Extract the relevant text concents and append it to the corpus
            for sdataset in graph.subjects(RDF.type, DCAT.Dataset):
                d           = dict(graph.predicate_objects(sdataset))
                title       = get_value(d, DCT.title)
                description = get_value(d, DCT.description)
```

```
print ('processing {}'.format(sdataset))
if title and description:
    text          = '.'.join([title, description])

    if bool(BeautifulSoup(text, "html5lib").find()):
        text = BeautifulSoup(text, "html5lib").text
    dataset_ref.append(sdataset)
    corpus.append(text)
```

```
processing http://78.91.98.234/dataset/5f7ec8a6-f727-44c8-aa8c-3cc4171c9d5c
processing http://78.91.98.234/dataset/a2f25723-7abc-4d9a-b1e4-1691e1f85839
processing http://78.91.98.234/dataset/d38ec913-559c-4eeb-8ee7-59fef63db90d
processing http://78.91.98.234/dataset/627b13f4-5f4b-4e44-b53e-e37dc6a70ac2
processing http://78.91.98.234/dataset/b618029c-3f30-4ea1-a2be-44387de5cb9d
processing http://78.91.98.234/dataset/f8bf6b37-c303-48b4-b7bd-61ebf94f9822
processing http://78.91.98.234/dataset/d46c7ad6-b2b1-4723-a31f-174070ab6401
processing http://78.91.98.234/dataset/efd73feb-7e7b-43fc-91b7-b9600579dbb6
processing http://78.91.98.234/dataset/ccf58a74-4079-402e-95bc-88baca35e981
processing http://78.91.98.234/dataset/5992974c-62b0-499b-9303-f54bf1f38b03
processing http://78.91.98.234/dataset/fa597582-7517-4428-8fb9-9057c8f818d8
processing http://78.91.98.234/dataset/e97fd128-4db5-4b0a-ab36-658c489b76eb
processing http://78.91.98.234/dataset/125f0f86-8cf5-4630-ae7e-c0fee28c0dac
processing http://78.91.98.234/dataset/baa21b11-e7e1-4724-9410-f13992e9d30a
processing http://78.91.98.234/dataset/14d06cbf-e21c-4d44-b940-43cee0aac2ba
processing http://78.91.98.234/dataset/5a482f2a-332b-470a-a846-44fa14a88530
processing http://78.91.98.234/dataset/5b7efc02-5864-4020-96e6-c3d81b13a06a
processing http://78.91.98.234/dataset/b6dd64eb-4857-4063-8b87-bba122776824
processing http://78.91.98.234/dataset/767baf63-e43a-4cab-ba6e-db3d30363ce2
processing http://78.91.98.234/dataset/be179b53-fe10-48cd-be1f-e4f6dc1b24e9
processing http://78.91.98.234/dataset/c1392a5e-675e-4e1a-9e79-ed4e0c17328a
processing http://78.91.98.234/dataset/6532c035-257d-4808-9883-38614e80fd04
processing http://78.91.98.234/dataset/0cd2de3b-e76f-49d8-b8cc-4c1013f59f85
processing http://78.91.98.234/dataset/29078456-a2e4-4659-bd37-b2c029182703
processing http://78.91.98.234/dataset/5e5bab6c-0659-4f95-b65e-85fb454f88a4
processing http://78.91.98.234/dataset/b58d8633-0a0c-438c-b283-c4b0ae13cd77
processing http://78.91.98.234/dataset/3433bcbe-e8cc-465f-bee4-95265a29b57b
processing http://78.91.98.234/dataset/a07a14eb-5b51-4484-ab0e-b8ec2bfdb2fa
processing http://78.91.98.234/dataset/4c13d555-fdfe-48b1-bfe6-db3c96c05a69
processing http://78.91.98.234/dataset/f6475ec0-099f-4cc8-84f7-4f6b3aaae393
processing http://78.91.98.234/dataset/0e4afab3-e679-4d3d-8052-0f793158cd7f
processing http://78.91.98.234/dataset/53a3c05b-973a-4f8d-9ec3-c2fddec576e0
processing http://78.91.98.234/dataset/1284a379-64f8-4066-bf2c-155a76c0a80f
processing http://78.91.98.234/dataset/a5b546fc-b9b1-4766-bd72-866e0edb874d
processing http://78.91.98.234/dataset/2adf0936-b181-4cd5-82f8-fff6802b2214
processing http://78.91.98.234/dataset/14027ce4-09de-4a1c-b2bc-61f5ab61bfd0
processing http://78.91.98.234/dataset/5e604a17-dab3-4e6a-977d-ff4964338296
processing http://78.91.98.234/dataset/ecb58c29-3169-4eef-8873-1483d1c5e88c
...
```

```
processing http://78.91.98.234/dataset/055880c9-cb7e-4919-ab9f-e6d6ee096346
processing http://78.91.98.234/dataset/16d3afdb-bb20-4593-929a-9fbd3f52ef05
processing http://78.91.98.234/dataset/ee2f48ab-5b04-4c80-ad3f-0c6cfb87cc27
processing http://78.91.98.234/dataset/8f8ac030-0d03-46e2-8eb7-844ee11a6203
processing http://78.91.98.234/dataset/7f6df84e-409c-4509-ba95-23a13d0a6730
processing http://78.91.98.234/dataset/6837c1de-6dce-48a3-a8a6-e59630912779
processing http://78.91.98.234/dataset/c1a060b6-350c-433d-ac78-964ae8b0a9e3
processing http://78.91.98.234/dataset/de487b7d-9030-4ca6-beed-3b0291e8b608
processing http://78.91.98.234/dataset/86d3fe44-111e-4d82-be5a-67a9dbfbfcbb
processing http://78.91.98.234/dataset/afae0c45-1c59-44b5-a433-a6c61da2cb2e
processing http://78.91.98.234/dataset/b9ca09a0-983c-4dc2-a76e-eb6b40a4888e
processing http://78.91.98.234/dataset/ca29e1ab-0545-4674-b970-abe922806294
processing http://78.91.98.234/dataset/0750ce3b-f383-45ad-80ee-eebb46c0fafe
processing http://78.91.98.234/dataset/36ceda99-bbc3-4909-bc52-b05a6d634b3f
processing http://78.91.98.234/dataset/50f95336-3be4-4994-837d-a1f098bcf105
processing http://78.91.98.234/dataset/376d51cf-450d-4d7a-b882-1d9e2b4f7b33
processing http://78.91.98.234/dataset/266084a5-b5b7-4993-9e8d-11eb14324269
processing http://78.91.98.234/dataset/f4c5e257-1c0e-4ff9-94d5-6088833fb19e
processing http://78.91.98.234/dataset/cb25bd3d-be11-4aff-af2d-a6415d417c42
processing http://78.91.98.234/dataset/0bb2463d-9564-452e-ae9e-245f9d341773
processing http://78.91.98.234/dataset/90cef5d5-601e-4412-87e4-3e9e8dc59245
processing http://78.91.98.234/dataset/0f0e037e-b5e8-453f-97ca-8ae9be7e523c
processing http://78.91.98.234/dataset/8113c645-840b-4cb3-adec-ffdcf7d1c07b
processing http://78.91.98.234/dataset/52546226-9fc9-4020-bd21-bd869c5ba8bd
processing http://78.91.98.234/dataset/d80802b2-695a-483c-8a6e-88eb0d961020
processing http://78.91.98.234/dataset/fb3cd5ed-c920-4369-aa84-e4a0a802499e
processing http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-085078634e01
processing http://78.91.98.234/dataset/aad23951-b6eb-489e-8bfe-59949eed916a
processing http://78.91.98.234/dataset/d7e6d74f-04a3-4b55-9f9e-a0d98003a666
processing http://78.91.98.234/dataset/bf627d4a-f115-41a2-82b9-d19de3cd5414
processing http://78.91.98.234/dataset/975491e3-bb7a-49bf-a575-7e75ffdca0bd
processing http://78.91.98.234/dataset/3863c79d-1102-45dc-aecb-6d0c0d0a1696
processing http://78.91.98.234/dataset/1510b486-ef6a-4c03-a858-b19a3a802f82
processing http://78.91.98.234/dataset/ba440808-277f-4fd1-9276-626e61434c11
processing http://78.91.98.234/dataset/6370c400-1fd2-45b0-a018-1362b7303088
processing http://78.91.98.234/dataset/81a17167-add0-4a24-8acc-75f1d87d6d66
processing http://78.91.98.234/dataset/1b85ba90-b675-4831-87fd-4d0de893df18
```

```
In [2]: from uuid import uuid4
        from nltk.corpus import stopwords
        from nltk.tokenize import sent_tokenize
        from nltk.tokenize import word_tokenize
        from sklearn.feature_extraction.text import TfidfVectorizer

        DCT = Namespace('http://purl.org/dc/terms/')

        norwegian_stop_words = [
    'og', 'i', 'jeg', 'det', 'at', 'en', 'et', 'den', 'til', 'er', 'som',
    'på', 'de', 'med', 'han', 'av', 'ikke', 'ikkje', 'der', 'så', 'var', 'meg',
    'seg', 'men', 'ett', 'har', 'om', 'vi', 'min', 'mitt', 'ha', 'hadde', 'hun',
    'nå', 'over', 'da', 'ved', 'fra', 'du', 'ut', 'sin', 'dem', 'oss', 'opp',
    'man', 'kan', 'hans', 'hvor', 'eller', 'hva', 'skal', 'selv', 'sjøl', 'her', 'alle',
    'vil', 'bli', 'ble', 'blei', 'blitt', 'kunne', 'inn', 'når', 'være', 'kom', 'noen',
    'noe', 'ville', 'dere', 'som', 'deres', 'kun', 'ja', 'etter', 'ned', 'skulle', 'denne',
    'for', 'deg', 'si', 'sine', 'sitt', 'mot', 'å', 'meget', 'hvorfor', 'dette', 'disse',
    'uten', 'hvordan', 'ingen', 'din', 'ditt', 'blir', 'samme', 'hvilken', 'hvilke', 'sånn', 'inni',
    'mellom', 'vår', 'hver', 'hvem', 'vors', 'hvis', 'både', 'bare', 'enn', 'fordi', 'før',
    'mange', 'også', 'slik', 'vært', 'være', 'båe', 'begge', 'siden', 'dykk', 'dykkar', 'dei',
    'deira', 'deires', 'deim', 'di', 'då', 'eg', 'ein', 'eit', 'eitt', 'elles', 'honom',
    'hjå', 'ho', 'hoe', 'henne', 'hennar', 'hennes', 'hoss', 'hossen', 'ikkje', 'ingi', 'inkje',
    'korleis', 'korso', 'kva', 'kvar', 'kvarhelst', 'kven', 'kvi', 'kvifor', 'me', 'medan', 'mi',
    'mine', 'mykje', 'no', 'nokon', 'noka', 'nokor', 'noko', 'nokre', 'si', 'sia', 'sidan',
    'so', 'somt', 'somme', 'um', 'upp', 'vere', 'vore', 'verte', 'vort', 'varte', 'vart',
    'alle', 'andre', 'arbeid', 'at', 'av', 'bare', 'begge', 'ble', 'blei', 'bli', 'blir',
    'blitt', 'bort', 'bra', 'bruke', 'både', 'båe', 'da', 'de', 'deg', 'dei', 'deim',
    'deira', 'deires', 'dem', 'den', 'denne', 'der', 'dere', 'deres', 'det', 'dette', 'di',
    'din', 'disse', 'ditt', 'du', 'dykk', 'dykkar', 'då', 'eg', 'ein', 'eit', 'eitt',
    'eller', 'elles', 'en', 'ene', 'eneste', 'enhver', 'enn', 'er', 'et', 'ett', 'etter',
    'folk', 'for', 'fordi', 'forsûke', 'fra', 'få', 'før', 'fûr', 'fûrst', 'gjorde', 'gjûre',
    'god', 'gå', 'ha', 'hadde', 'han', 'hans', 'har', 'hennar', 'henne', 'hennes', 'her',
    'hjå', 'ho', 'hoe', 'honom', 'hoss', 'hossen', 'hun', 'hva', 'hvem', 'hver', 'hvilke',
    'hvilken', 'hvis', 'hvor', 'hvordan', 'hvorfor', 'i', 'ikke', 'ikkje', 'ingen', 'ingi', 'inkje',
    'inn', 'innen', 'inni', 'ja', 'jeg', 'kan', 'kom', 'korleis', 'korso', 'kun', 'kunne',
    'kva', 'kvar', 'kvarhelst', 'kven', 'kvi', 'kvifor', 'lage', 'lang', 'lik', 'like', 'makt',
    'man', 'mange', 'me', 'med', 'medan', 'meg', 'meget', 'mellom', 'men', 'mens', 'mer',
    'mest', 'mi', 'min', 'mine', 'mitt', 'mot', 'mye', 'mykje', 'må', 'måte', 'navn',
    'ned', 'nei', 'no', 'noe', 'noen', 'noka', 'noko', 'nokon', 'nokor', 'nokre', 'ny',
    'nå', 'når', 'og', 'også', 'om', 'opp', 'oss', 'over', 'part', 'punkt', 'på',
    'rett', 'riktig', 'samme', 'sant', 'seg', 'selv', 'si', 'sia', 'sidan', 'siden', 'sin',
    'sine', 'sist', 'sitt', 'sjøl', 'skal', 'skulle', 'slik', 'slutt', 'so', 'som', 'somme',
    'somt', 'start', 'stille', 'så', 'sånn', 'tid', 'til', 'tilbake', 'tilstand', 'um', 'under',
    'upp', 'ut', 'uten', 'var', 'vart', 'varte', 'ved', 'verdi', 'vere', 'verte', 'vi',
    'vil', 'ville', 'vite', 'vore', 'vors', 'vort', 'vår', 'være', 'vært', 'vöre', 'å']

#tf = TfidfVectorizer(stop_words=norwegian_stop_words, use_idf=True,ngram_range=(1,3))
#tfidf_matrix = tf.fit_transform(corpus)
#feature_names = tf.get_feature_names()
#
#def weighted_keywords(idx, n_results=10):
#    feature_index = tfidf_matrix[idx,:].nonzero()[1]
#    tfidf_scores = zip(feature_index, [tfidf_matrix[idx, x] for x in feature_index])
#    pairs = [(w, s) for w, s in [(feature_names[i], s) for (i, s) in tfidf_scores]]
#    for pair in sorted(pairs, key=lambda v: v[1], reverse=True)[:n_results]:
#        yield pair
```

```
In [3]: # Import the norwegian WordNet "OrdVev".
        # Not that this is extremely slow
        from odt.ordvev import OrdVev
        ordvev = OrdVev()

In [4]: # Read ontology from database
        from odt.database import load_ontology
        config = {}
        config['DB_USERNAME'] = 'nims'
        config['DB_PASSWD'] = 'Da90hfw'
        config['ONTOLOGY_UUID'] = '5b2ab51401d5412566cf4e94'

        uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format(config['DB_USERN
                                                                   config['DB_PASSWD'])
        ontology_graph = load_ontology(uri, config['ONTOLOGY_UUID'])
        ontology_graph.serialize(format='turtle')

Out[4]: b'@prefix dcat: <http://www.w3.org/ns/dcat#> .\n@prefix dct: <http://purl.org/dc/terms/>
<http://www.quaat.com/ontologies#> .\n@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-n
In [5]: concepts = [c for c in ontology_graph.subjects(RDF.type, SKOS.Concept)]
        print (concepts)
        lblmap = {}
        # Extract all labels from a given concept
        # Extract all labels from a given concept
        for concept in concepts:
                pref = [p.value for p in ontology_graph.objects(concept, SKOS.prefLabel) if p.lan
                alt = [p.value for p in ontology_graph.objects(concept, SKOS.altLabel) if p.langu
                lblmap[concept] = pref+alt

        def remove_stopwords(tokens, stop_words):
            return [w for w in tokens if w not in stop_words]

        def normalize(tokens):
            return [w.lower() for w in tokens if w.isalnum() ]

        def compute_score(word, concept):
            score = 0.0
            for lbl in lblmap[concept]:
                new_score = ordvev.sim_wup(word, lbl)
                score = max(score, new_score)
            return score

        # Helper method to create a link between a dataset and a concept with
        # a given similarity-score
        def add_similarity_link(graph, dataset, concept, score):
            uuid = uuid4().hex
            simlink = URIRef(QEX[uuid])
            graph.add((simlink, RDF.type, ODT.Similarity))
```

```
    graph.add((simlink, ODT.dataset, dataset))
    graph.add((simlink, ODT.concept, concept))
    graph.add((simlink, ODT.score, Literal(score, datatype=XSD.double)))
    return simlink


wt = word_tokenize
#kolumbus_dataset = URIRef('http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f
#index = dataset_ref.index(kolumbus_dataset)

semantic_threshold = 0.8 #
min_concepts = 4# The minimum number of linked concepts

similarity_graph = Graph()
similarity_graph.bind('odt', ODT)

num_datasets = len(dataset_ref)
for index, dataset in enumerate(dataset_ref):
    print ('processing dataset {} of {}'. format(index, num_datasets))
    tokens = wt(corpus[index])
    tokens = normalize(remove_stopwords(tokens, norwegian_stop_words))
    #print (tokens)

    # Compute the relevance for each concept
    scorelist = []
    for concept in concepts:
        score = 0.0
        for token in tokens:
            new_score = compute_score(token, concept)
            score = max(new_score, score)
        scorelist.append((concept, score))
    sorted_scores = sorted(scorelist, key=lambda x: x[1], reverse=True)
    filtered_score = [(c,s) for c,s in sorted_scores if s >= semantic_threshold]
    if len(filtered_score) < 4:
        filtered_score = sorted_scores[:min_concepts]

    # Add the scores to a graph
    for concept, score in filtered_score:
        add_similarity_link(similarity_graph, dataset, concept, score)

similarity_graph.serialize(destination='autotag-no-v5.0.rdf', format='xml')
```

C Appendix

CCS Comparison - English Norwegian

July 8, 2018

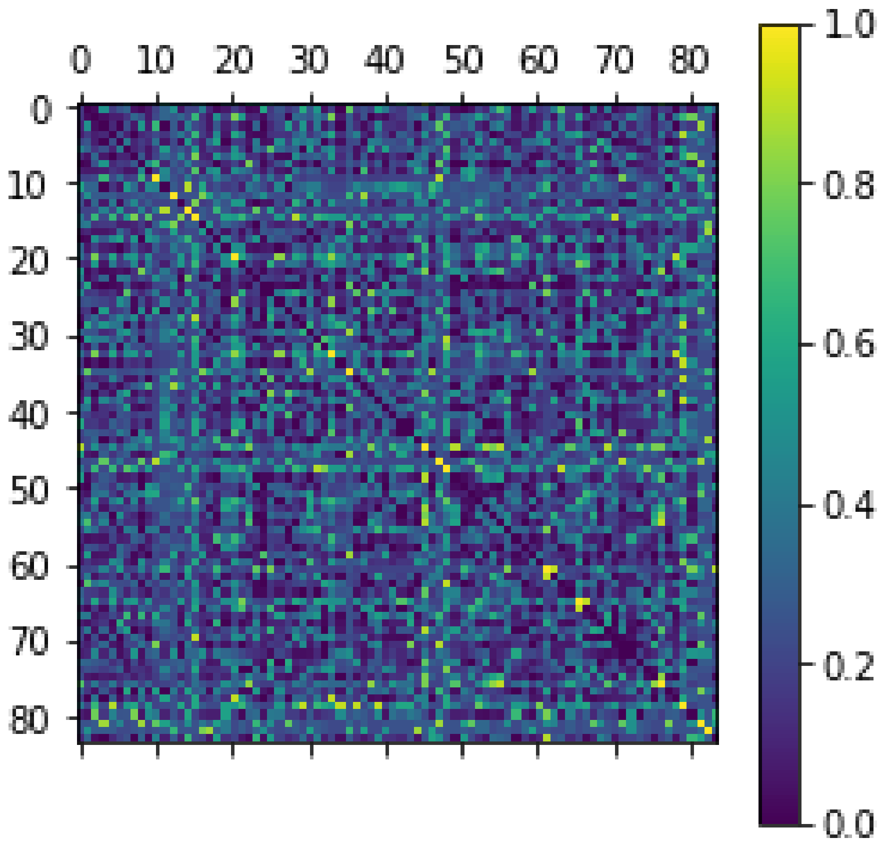
```
In [1]: from odt.ordvev import OrdVev
        ordvev = OrdVev()

In [3]: import numpy as np
        from rdflib import Graph
        from rdflib import Namespace
        from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD, SKOS
        from odt.database import load_ontology, load_dataset, load_similarity, load_autotagge
        from odt.opendatasemanticframework import OpenDataSemanticFramework

        ONTOLOGY_UUID = '5b0da4de01d541154b719008'
        uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw')

        ODT = Namespace('http://www.quaat.com/ontologies#')
        DCT = Namespace('http://purl.org/dc/terms/')

        ontology_graph = load_ontology(uri,ONTOLOGY_UUID)
        ontology_graph.bind('dct', DCT)
        ontology_graph.bind('owl', OWL)
        ontology_graph.bind('rdf', RDF)
        ontology_graph.bind('rdfs', RDFS)
        ontology_graph.bind('foaf', FOAF)
        ontology_graph.bind('dc', DC)
        ontology_graph.bind('odt', ODT)
        ontology_graph.bind('skos', SKOS)

In [18]: import pandas as pd
         data = []
         lblmap = {}
         concepts = list(ontology_graph.subjects(RDF.type, SKOS.Concept))

         for concept in concepts:
             pref = [p.value for p in ontology_graph.objects(concept, SKOS.prefLabel) if p.la
             alt = [p.value for p in ontology_graph.objects(concept, SKOS.altLabel) if p.langu
             lblmap[concept] = pref+alt

         def compare_concepts(lblmap, concept1, concept2):
```

```
        score = 0.0
        for w1 in lblmap[concept1]:
            for w2 in lblmap[concept2]:
                score = max(ordvev.sim_wup(w1,w2), score)
        return score


    for c1 in concepts:
        v = []
        for c2 in concepts:
            score = compare_concepts(lblmap,c1,c2)
            v.append(score)
        data.append(v)
    ccs = pd.DataFrame(columns=concepts, index=concepts, data=data)
    ccs_en = load_dataframe(uri,'5b02df9101d54140f183edb2')
    abs(ccs_en - ccs)
```

```
Out[18]:                                                      http://www.quaat.com/ontologies#
    http://www.quaat.com/ontologies#Abstraction                     0.000000
    http://www.quaat.com/ontologies#Accident                        0.063492
    http://www.quaat.com/ontologies#Action                          0.083333
    http://www.quaat.com/ontologies#Activity                        0.083333
    http://www.quaat.com/ontologies#Agreement                       0.466667
    http://www.quaat.com/ontologies#Airport                         0.063492
    http://www.quaat.com/ontologies#Artifact                        0.150000
    http://www.quaat.com/ontologies#Bicycle                         0.068182
    http://www.quaat.com/ontologies#Boat                            0.027778
    http://www.quaat.com/ontologies#Booth                           0.085714
    http://www.quaat.com/ontologies#Bridge                          0.063492
    http://www.quaat.com/ontologies#Building                        0.085714
    http://www.quaat.com/ontologies#Bus                             0.050000
    http://www.quaat.com/ontologies#BusTerminal                     0.068182
    http://www.quaat.com/ontologies#ChargingStation                 0.250000
    http://www.quaat.com/ontologies#Collection                      0.380952
    http://www.quaat.com/ontologies#Communication                   0.600000
    http://www.quaat.com/ontologies#Company                         0.126984
    http://www.quaat.com/ontologies#Detour                          0.027778
    http://www.quaat.com/ontologies#Emission                        0.133333
    http://www.quaat.com/ontologies#Entity                          0.666667
    http://www.quaat.com/ontologies#EnvironmentalIn...              0.126984
    http://www.quaat.com/ontologies#Event                           0.150000
    http://www.quaat.com/ontologies#Facility                        0.111111
    http://www.quaat.com/ontologies#Forecast                        0.285714
    http://www.quaat.com/ontologies#Formation                       0.444444
    http://www.quaat.com/ontologies#Garage                          0.083333
    http://www.quaat.com/ontologies#Group                           0.514286
    http://www.quaat.com/ontologies#Happening                       0.083333
    http://www.quaat.com/ontologies#Highway                         0.083333
```

1 Comparing english and norwegian Concept-Concept Similarity Scores

In [13]:
```
from odt.database import load_dataframe
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

diff = abs(ccs_en.as_matrix()-ccs.as_matrix())
ccs_diff = pd.DataFrame(columns=concepts, index=concepts, data=diff)
plt.matshow(diff)
plt.colorbar()
plt.show()
```



D Appendix

Save Datasets &
Dataset Analysis

July 9, 2018

1 Storing graphs on the online database storage server

```
In [1]: from rdflib import Graph
        from rdflib import URIRef, Graph, BNode, Literal
        from rdflib import Namespace
        from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD, SKOS
        from odt.skosnavigate import SKOSNavigate
        from uuid import uuid4
        from pymongo import MongoClient
        import pandas as pd
        import numpy as np

        ODT = Namespace('http://www.quaat.com/ontologies#')
        DCAT = Namespace('http://www.w3.org/ns/dcat#')
        DCT = Namespace('http://purl.org/dc/terms/')
        ODTX = Namespace('http://www.quaat.com/ontology/ODTX#')
        QEX = Namespace('http://www.quaat.com/extended_skos#')
        VCARD = Namespace('http://www.w3.org/2006/vcard/ns#')
        WN20SCHEMA = Namespace('http://www.w3.org/2006/03/wn/wn20/schema/')
        DN = Namespace('http://www.wordnet.dk/owl/instance/2009/03/instances/')
        DN_SCHEMA = Namespace('http://www.wordnet.dk/owl/instance/2009/03/schema/')

In [ ]: graph = Graph()
        ODT = Namespace('http://www.quaat.com/ontologies#')
        graph.bind('odt', ODT)
        with open('satellittdata.json') as file:
            data = file.read()
            graph.parse(data=data, format='json-ld')

        graph.serialize()

In [16]: graph = Graph()
         graph.parse('otd-ontology-v5.owl', format='xml')
         jld = graph.serialize(format='json-ld')
         uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw
         client = MongoClient(uri)
         db = client.ontodb
         onts = db.ontologies
        ont = {"label": "test", "ontology": jld}
        ontId = onts.insert_one(ont).inserted_id
        print (str(ontId))

5b2ab51401d5412566cf4e94


In [6]: graph = Graph()
        graph.parse('all_output.rdf', format='xml')
        jld = graph.serialize(format='json-ld')
        uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw')
        client = MongoClient(uri)
        db = client.ontodb
        onts = db.ontologies
        ont = {"label": "datasets", "datasets": jld}
        ontId = onts.insert_one(ont).inserted_id
        print (str(ontId))

5b2968c501d5412566cf4e86


In [23]: graph = Graph()
         graph.parse('tagged-v0.8.json', format='json-ld')
         jld = graph.serialize(format='json-ld')
         uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw
         client = MongoClient(uri)
         db = client.ontodb
         onts = db.ontologies
         ont = {"label": "tagged", "similarity": jld}
         ontId = onts.insert_one(ont).inserted_id
         print (str(ontId))

5b2ada4d01d5412566cf4ea1


In [19]: # Store automatic tagging scheme
         graph = Graph()
         graph.parse('autotag-no-v5.0.rdf', format='xml')
         jld = graph.serialize(format='json-ld')
         uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw
         client = MongoClient(uri)
         db = client.ontodb
         onts = db.ontologies
         ont = {"label": "autotagged", "autotag": jld}
         ontId = onts.insert_one(ont).inserted_id
         print (str(ontId))

5b2acdfe01d5412566cf4e99
```

```
In [ ]: from odt.database import load_autotagged
        from rdflib import Namespace
        from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD, SKOS
        ODT = Namespace('http://www.quaat.com/ontologies#')
        DCAT = Namespace('http://www.w3.org/ns/dcat#')
        DCT = Namespace('http://purl.org/dc/terms/')

        g = load_autotagged(uri, str(ontId))
        g.bind('odt', ODT)
        g.bind('dcat', DCAT)
        g.bind('dct', DCT)
        g.serialize()
```

```
In [ ]: from bson.objectid import ObjectId
        client = MongoClient(uri)
        db = client.ontodb
        onts = db.ontologies
        doc = onts.find_one({'_id': ObjectId(ontId)})
        d = doc['ontology'].decode("utf-8")
        graph = Graph()
        graph.parse(data=d, format='json-ld')
        #graph.serialize()
```

```
In [ ]: import requests
        package_list = requests.get('http://78.91.98.234:5000/api/3/action/package_list')
        package_list.status_code
        graph = Graph()
        for res in package_list.json()['result']:
            dataset='http://78.91.98.234:5000/dataset/{0}.rdf'.format(res)
            data = requests.get(dataset)
            graph.parse(data=data.text)
        graph.serialize(destination='all_output.rdf', format='xml')
```

2 Dataset analysis.

This section is copied from the "Extract concepts with preferred and alternative labels"

```
In [18]:  # Fetching the ontology from the database
          from odt.database import load_ontology
          from odt.ordvev import OrdVev

          ontology_uuid = '5b2ab51401d5412566cf4e94'
          uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('nims', 'Da90hfw

          graph = load_ontology(uri, ontology_uuid)
          #graph.serialize(destination='5afc0fb801d5415a9d324003.rdf', format='xml')

          ordvev = OrdVev()
```

```
         # List english concepts and dictionary concepts
         from nltk.corpus import wordnet as wn

In [11]: num_concepts = 0
         num_no_preflabel_in_dict = 0
         num_en_preflabel_in_dict = 0
         num_no_altlabel_in_dict = 0
         num_en_altlabel_in_dict = 0
         num_en_covered = 0
         num_nb_covered = 0

         def synset_exists(tok):
             if len(wn.synsets(tok)) > 0:
                 return True
             return False

         def first(xs):
             if xs:
                 return xs[0]
             return None

         undefined_en_vocab = []
         undefined_en_alt_vocab = []
         undefined_vocab = []
         undefined_alt_vocab = []

         for concept in graph.subjects(RDF.type, SKOS.Concept):
             en_covered = False
             nb_covered = False
             num_concepts += 1
             for pref in [l for l in graph.objects(concept, SKOS.prefLabel) if l.language == 'en']:
                 if wn.synsets(pref):
                     num_en_preflabel_in_dict += 1
                     en_covered = True
                     break
                 else:
                     undefined_en_vocab.append(pref)

             for alt in [l for l in graph.objects(concept, SKOS.altLabel) if l.language == 'en']:
                 if wn.synsets(alt):
                     num_en_altlabel_in_dict += 1
                     en_covered = True
                     break
                 else:
                     undefined_en_alt_vocab.append(alt)

             for pref in [l for l in graph.objects(concept, SKOS.prefLabel) if l.language == 'nb']:
                 if list(ordvev.synsets(pref)):
```

```
                num_no_preflabel_in_dict += 1
                nb_covered = True
                break
            else:
                undefined_vocab.append(pref)

        for alt in [l for l in graph.objects(concept, SKOS.altLabel) if l.language == 'nb']:
            if list(ordvev.synsets(alt)):
                num_no_altlabel_in_dict += 1
                nb_covered = True
                break
            else:
                undefined_alt_vocab.append(alt)

        if en_covered:
            num_en_covered += 1

        if nb_covered:
            num_nb_covered += 1

    print ('num concepts: ', num_concepts)
    print ('num english preflabels in dictionary: {0} ({1}%)'.format(num_en_preflabel_in_dic
    print ('num english altlabels in dictionary: {0} ({1}%)'.format(num_en_altlabel_in_dict, 1
    print ('num norwegian preflabels in dictionary: {0} ({1}%)'.format(num_no_preflabel_in_
    print ('num norwegian altlabels in dictionary: {0} ({1}%)'.format(num_no_altlabel_in_dic
    print ('num english concepts covered: {0} ({1}%)'.format(num_en_covered, 100*num_e
    print ('num norwegian concepts covered: {0} ({1}%)'.format(num_nb_covered, 100*nu

    print ("\n=== Undefined pref 'en' tokens")
    for voc in undefined_en_vocab:
        print (str(voc))

    print ("\n=== Undefined alternative 'en' tokens")
    for voc in undefined_en_alt_vocab:
        print (str(voc))

    print ("\n=== Undefined pref 'nb' tokens")
    for voc in undefined_vocab:
        print (str(voc))

    print ("\n=== Undefined alternative 'nb' tokens")
    for voc in undefined_alt_vocab:
        print (str(voc))
```

```
num concepts:  102
num english preflabels in dictionary: 67 (65.68627450980392%)
num english altlabels in dictionary: 4 (3.9215686274509802%)
num norwegian preflabels in dictionary: 72 (70.58823529411765%)
num norwegian altlabels in dictionary: 9 (8.823529411764707%)
num english concepts covered: 71 (69.6078431372549%)
num norwegian concepts covered: 75 (73.52941176470588%)

=== Undefined pref 'en' tokens
travel plan
personal transport
topographic point
public transport
traffic condition
point of interest
geographical information
weather forecast
street sign
service area
real time
bus stop
emission information
charging station
traffic information
environment information
route plan
traffic flow
railway junction
hydrometeorological information
bus terminal
air quality
traffic queue
park and ride
weather condition
meteorological information
traffic circle
transport mode
API description
transfer node
transport network condition
tracking information
travel information
t junction
tool booth

=== Undefined alternative 'en' tokens
bus station
coaching inn
transfer point
3-way junction

=== Undefined pref 'nb' tokens
```

landevei
persolig transport
topografisk punkt
offentlig transport
trafikkforhold
interessepunkt
geografisk informasjon
trafikkskilt
serviceområde
sanntid
ladestasjon
trafikkinformasjon
miljøinformasjuton
ruteplan
trafikkflyt
hydrometeorologisk informasjon
entitet
skysstasjon
meteorologisk informasjon
skinner
rundkjøring
transporttype
overføringspunkt
transportnettforhold
sporing
prediksjon
reiseinformation
3-veis kryss
bomstasjon

=== Undefined alternative 'nb' tokens
trasse
skyss
geoinformasjon
discharge
utslippsdata
lading

landevei - fylkesvei meteorologisk informasjon - meteoriologi transportnettilstand topografisk punkt trafikkforhold - trafikkproblem persolig transport trafikkø serviceområde trafikkinformasjon offentlig transport - kollektivtransport bomstasjon - veiavgift(?) entitet trafikkskilt - veiskilt rundkjøring ruteplan ladestasjon 3-veis kryss - gatekryss trafikkflyt - trafikkstrøm overføringspunkt

E Appendix

manual tagging

July 9, 2018

1 Create Similarity links from Manual tagging

This code is performing "manuall tagging" by creating similary links as defined in the in document "Manually tagged open transport data", (Hagelien,Jiang, & Natvig, 2018)

In [2]:
```
from rdflib import Namespace
from rdflib import BNode
from rdflib.namespace import RDF, RDFS, OWL, DC, FOAF, XSD, SKOS
from rdflib import plugin, Graph, Literal, URIRef
from uuid import uuid4

graph = Graph()

OTD = Namespace('http://www.quaat.com/ontologies#')
DCAT = Namespace('http://www.w3.org/ns/dcat#')
DCT = Namespace('http://purl.org/dc/terms/')
ODTX = Namespace('http://www.quaat.com/ontology/ODTX#')
QEX = Namespace('http://www.quaat.com/extended_skos#')
WN20SCHEMA = Namespace('http://www.w3.org/2006/03/wn/wn20/schema/')
DN = Namespace('http://www.wordnet.dk/owl/instance/2009/03/instances/')
DN_SCHEMA = Namespace('http://www.wordnet.dk/owl/instance/2009/03/schema/')


graph.bind('otd', OTD)
graph.bind('qex', QEX)

# Helper method to create a link between a dataset and a concept with
# a given similarity-score
def add_similarity_link(graph, dataset, concept, score=1.0):
    uuid = uuid4().hex
    simlink = URIRef(QEX[uuid])
    graph.add((simlink, RDF.type, OTD.Similarity))
    graph.add((simlink, OTD.dataset, dataset))
    graph.add((simlink, OTD.concept, concept))
    graph.add((simlink, OTD.score, Literal(score, datatype=XSD.double)))
    return simlink
```

```
# Define a list of dataset-concept pairs that corresponds
# to the document "manually tagged  open transport data"
dataset_concept_pairs = [
    ('caefad21-41cc-41e6-9f59-e050486686ea', OTD.ChargingStation),
    ('88d19fa6-3c12-4838-b58e-38d28e860245', OTD.AirQuality),
    ('d473bc01-8d6e-4771-a946-e19d3fff3691', OTD.AirQuality),
    ('1f64a769-9c10-4cc7-9db9-60ac74a7183e', OTD.Location),
    ('3e661b0d-d325-4ca4-b536-972b69615629', OTD.Location),
    ('9f6d9caa-df3a-4515-8ba3-025c076efaeb', OTD.HydrometeorologicalInformation),
    ('055880c9-cb7e-4919-ab9f-e6d6ee096346', OTD.Location),
    ('0612abbf-1b35-422c-971c-869c6f22214e', OTD.Location),
    ('0827894d-5db2-443b-b8fb-e58e60cb6962', OTD.Location),
    ('12bf3448-ec94-4188-8a1f-c89255e6217c', OTD.Location),
    ('1a7f9e36-0e6f-41e6-9608-4f184e27c5ed', OTD.Location),
    ('4298b3a3-d06e-40bc-9097-4229e993988e', OTD.Location),
    ('4fbaaeb9-b6bb-4862-b8cf-aee476725d1f', OTD.Location),
    ('64a3f8ff-5ce8-40f7-bf9d-33e335d406ce', OTD.Location),
    ('6adb0d93-6cb3-4e4b-861f-489f3b9bd9b2', OTD.Location),
    ('6e24e79e-3bda-4a65-87fb-8123f5bb1cde', OTD.Location),
    ('85cd5905-a6a0-422b-ae8c-776b9f84d205', OTD.Location),
    ('8966a9f8-c4aa-4c83-ae1c-b9de223a728d', OTD.Location),
    ('89cf5416-caf8-46d7-9d2d-7bf57ea9d1ac', OTD.Location),
    ('8f8ac030-0d03-46e2-8eb7-844ee11a6203', OTD.Location),
    ('94016332-2c3f-4c5d-9798-4582961ead79', OTD.Location),
    ('9c18dcad-313d-4bb0-bec7-358c40724cf0', OTD.Location),
    ('a68c8302-5192-4a34-9134-98500dd44f74', OTD.Location),
    ('a8669394-1dc6-4258-be23-09f671d7c513', OTD.Location),
    ('a8cb46ae-7441-4a87-b52e-5351685a33d0', OTD.Location),
    ('ad9335f3-a405-4811-86a2-e7cb34254e11', OTD.Location),
    ('b6feae31-74cd-4ada-bc44-0a372f518380', OTD.Location),
    ('c028cc71-066c-42d6-8e66-31752b2d3a48', OTD.Location),
    ('d1a3a50b-0566-48c1-acc0-15049da971b3', OTD.Location),
    ('de82c50b-52c2-4266-9765-386a32c5302d', OTD.Location),
    ('dfb9b81c-d9a2-4542-8f63-7584a3594e02', OTD.Location),
    ('eca0854c-32dd-4692-8caa-dbdbd41fd95a', OTD.Location),
    ('ee2f48ab-5b04-4c80-ad3f-0c6cfb87cc27', OTD.Location),
    ('f5b758cf-9a21-419a-a311-e6eceeb7f29c', OTD.Location),
    ('36ceda99-bbc3-4909-bc52-b05a6d634b3f', OTD.Location),
    ('36ceda99-bbc3-4909-bc52-b05a6d634b3f', OTD.Parking),
    ('52d0d63e-d26c-43b9-91e2-13b3b2a96c89', OTD.Location),
    ('52d0d63e-d26c-43b9-91e2-13b3b2a96c89', OTD.Parking),
    ('95797234-3d9b-415c-a7e2-c4660843ee20', OTD.Location),
    ('95797234-3d9b-415c-a7e2-c4660843ee20', OTD.Parking),
    ('a487ff09-435a-44bf-8c35-36c60b1eb4ad', OTD.Location),
    ('a487ff09-435a-44bf-8c35-36c60b1eb4ad', OTD.TravelInformation),
    ('23fef01e-c729-43b2-8fb3-8e127f04b286', OTD.Map),
    ('0f0e037e-b5e8-453f-97ca-8ae9be7e523c', OTD.Map),
    ('0f0e037e-b5e8-453f-97ca-8ae9be7e523c', OTD.Bicycle),
('0f0e037e-b5e8-453f-97ca-8ae9be7e523c', OTD.TransportNetwork),
('bf374070-430b-4d30-8818-ff5ca756a3da', OTD.MeteorologicalInformation),
('1510b486-ef6a-4c03-a858-b19a3a802f82', OTD.MeteorologicalInformation),
('4698e844-2cc0-429a-958e-0853904e8652', OTD.Statistics),
('e97fd128-4db5-4b0a-ab36-658c489b76eb', OTD.Statistics),
('f2c97ee0-6cad-4833-92fb-4a21d8b68b74', OTD.Statistics),
('8feb944e-b50d-4536-a61a-cbdcf52861f1', OTD.Statistics),
('8feb944e-b50d-4536-a61a-cbdcf52861f1', OTD.APIDescription),
('ab60e637-ef33-44ea-b1d4-0bb0fea81799', OTD.Statistics),
('ab60e637-ef33-44ea-b1d4-0bb0fea81799', OTD.EnvironmentInformation),
('027b3371-5d84-4ac4-8dda-bba833189db7', OTD.TrafficInformation),
('027b3371-5d84-4ac4-8dda-bba833189db7', OTD.Bicycle),
('90cef5d5-601e-4412-87e4-3e9e8dc59245', OTD.TrafficInformation),
('90cef5d5-601e-4412-87e4-3e9e8dc59245', OTD.Bicycle),
('42fb5664-7801-4787-9d93-7b0c87fad360', OTD.TrafficInformation),
('42fb5664-7801-4787-9d93-7b0c87fad360', OTD.Bicycle),
('9e449b4a-a7c9-4d47-9d5e-94128d179995', OTD.TrafficInformation),
('9e449b4a-a7c9-4d47-9d5e-94128d179995', OTD.Road),
('52546226-9fc9-4020-bd21-bd869c5ba8bd', OTD.TrafficInformation),
('52546226-9fc9-4020-bd21-bd869c5ba8bd', OTD.Bicycle),
('aad23951-b6eb-489e-8bfe-59949eed916a', OTD.TrafficInformation),
('aad23951-b6eb-489e-8bfe-59949eed916a', OTD.Road),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.TrafficInformation),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.Road),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.RealTime),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.MeteorologicalInformation),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.TransportNetworkCondition),
('8c920c9b-77ad-4f64-8253-085078634e01', OTD.Accident),
('fb3cd5ed-c920-4369-aa84-e4a0a802499e', OTD.TrafficInformation),
('fb3cd5ed-c920-4369-aa84-e4a0a802499e', OTD.Train),
('8f3b2912-a296-42f9-bd3c-afeea8c678ab', OTD.Law),
('482b30a8-1b35-4ed0-85ec-cc52381fa422', OTD.TransferNode),
('482b30a8-1b35-4ed0-85ec-cc52381fa422', OTD.BusStop),
('08c25788-f90a-452c-bde0-1511cbbeddb8', OTD.TransportNetwork),
('08c25788-f90a-452c-bde0-1511cbbeddb8', OTD.Trail),
('0a4c57ac-386e-4a9b-ad09-c3d17c6a0725', OTD.TransportNetwork),
('0a4c57ac-386e-4a9b-ad09-c3d17c6a0725', OTD.Trail),
('0cd2de3b-e76f-49d8-b8cc-4c1013f59f85', OTD.TransportNetwork),
('0cd2de3b-e76f-49d8-b8cc-4c1013f59f85', OTD.Trail),
('0e4afab3-e679-4d3d-8052-0f793158cd7f', OTD.TransportNetwork),
('0e4afab3-e679-4d3d-8052-0f793158cd7f', OTD.Trail),
('1284a379-64f8-4066-bf2c-155a76c0a80f', OTD.TransportNetwork),
('1284a379-64f8-4066-bf2c-155a76c0a80f', OTD.Trail),
('1335971d-58ad-4a95-9b60-a1221636c3e0', OTD.TransportNetwork),
('1335971d-58ad-4a95-9b60-a1221636c3e0', OTD.Trail),
('14027ce4-09de-4a1c-b2bc-61f5ab61bfd0', OTD.TransportNetwork),
('14027ce4-09de-4a1c-b2bc-61f5ab61bfd0', OTD.Trail),
('1a3f7855-45f4-473c-89af-7fda8eede930', OTD.TransportNetwork),
('1a3f7855-45f4-473c-89af-7fda8eede930', OTD.Trail),
('221b490a-b9cc-4430-9bf1-ad59c4bbe06d', OTD.TransportNetwork),
('221b490a-b9cc-4430-9bf1-ad59c4bbe06d', OTD.Trail),
('22a016c9-c87e-4c14-b955-0b8f813773d9', OTD.TransportNetwork),
('22a016c9-c87e-4c14-b955-0b8f813773d9', OTD.Trail),
('25a7128a-1daf-485a-967e-dc208d0fa757', OTD.TransportNetwork),
('25a7128a-1daf-485a-967e-dc208d0fa757', OTD.Trail),
('29078456-a2e4-4659-bd37-b2c029182703', OTD.TransportNetwork),
('29078456-a2e4-4659-bd37-b2c029182703', OTD.Trail),
('2adf0936-b181-4cd5-82f8-fff6802b2214', OTD.TransportNetwork),
('2adf0936-b181-4cd5-82f8-fff6802b2214', OTD.Trail),
('2bfa7959-e9bc-4f4c-b1fa-46babdb08f42', OTD.TransportNetwork),
('2bfa7959-e9bc-4f4c-b1fa-46babdb08f42', OTD.Trail),
('2e78cdcc-897f-42d7-bd78-a09e6e070457', OTD.TransportNetwork),
('2e78cdcc-897f-42d7-bd78-a09e6e070457', OTD.Trail),
('3433bcbe-e8cc-465f-bee4-95265a29b57b', OTD.TransportNetwork),
('3433bcbe-e8cc-465f-bee4-95265a29b57b', OTD.Trail),
('42e225d3-4bf0-45ca-b579-500c62ad1340', OTD.TransportNetwork),
('42e225d3-4bf0-45ca-b579-500c62ad1340', OTD.Trail),
('48b50420-74c4-4c0c-9b60-20f203a41a06', OTD.TransportNetwork),
('48b50420-74c4-4c0c-9b60-20f203a41a06', OTD.Trail),
('49f3ebf9-40a8-4dd8-bd98-e7a0cee9a561', OTD.TransportNetwork),
('49f3ebf9-40a8-4dd8-bd98-e7a0cee9a561', OTD.Trail),
('4a8324d4-f531-41bc-8964-2dc998c05121', OTD.TransportNetwork),
('4a8324d4-f531-41bc-8964-2dc998c05121', OTD.Trail),
('4ae81d47-ca9d-4dd8-a65c-47b5e63bf427', OTD.TransportNetwork),
('4ae81d47-ca9d-4dd8-a65c-47b5e63bf427', OTD.Trail),
('4c13d555-fdfe-48b1-bfe6-db3c96c05a69', OTD.TransportNetwork),
('4c13d555-fdfe-48b1-bfe6-db3c96c05a69', OTD.Trail),
('4cee70c8-2fdc-4d25-a8e7-3daa2d7a376d', OTD.TransportNetwork),
('4cee70c8-2fdc-4d25-a8e7-3daa2d7a376d', OTD.Trail),
('51f6952b-069d-4af8-9a43-425ec054e41d', OTD.TransportNetwork),
('51f6952b-069d-4af8-9a43-425ec054e41d', OTD.Trail),
('53a3c05b-973a-4f8d-9ec3-c2fddec576e0', OTD.TransportNetwork),
('53a3c05b-973a-4f8d-9ec3-c2fddec576e0', OTD.Trail),
('584295a1-248c-4a35-b8d5-3fa2d71de880', OTD.TransportNetwork),
('584295a1-248c-4a35-b8d5-3fa2d71de880', OTD.Trail),
('5bb5d4d6-57db-4bb8-9c0b-e1e43bb15e4b', OTD.TransportNetwork),
('5bb5d4d6-57db-4bb8-9c0b-e1e43bb15e4b', OTD.Trail),
('5e5bab6c-0659-4f95-b65e-85fb454f88a4', OTD.TransportNetwork),
('5e5bab6c-0659-4f95-b65e-85fb454f88a4', OTD.Trail),
('5e604a17-dab3-4e6a-977d-ff4964338296', OTD.TransportNetwork),
('5e604a17-dab3-4e6a-977d-ff4964338296', OTD.Trail),
('64938d2c-859b-4370-8f58-9dce554776ae', OTD.TransportNetwork),
('64938d2c-859b-4370-8f58-9dce554776ae', OTD.Trail),
('650f6dc3-7d13-4472-a83a-3571a38db621', OTD.TransportNetwork),
('650f6dc3-7d13-4472-a83a-3571a38db621', OTD.Trail),
('6532c035-257d-4808-9883-38614e80fd04', OTD.TransportNetwork),
('6532c035-257d-4808-9883-38614e80fd04', OTD.Trail),
('725a493d-2942-4585-b72f-df6d1b06bb45', OTD.TransportNetwork),
('725a493d-2942-4585-b72f-df6d1b06bb45', OTD.Trail),
('92a9b284-6073-47e9-8eee-5f2195d0879a', OTD.TransportNetwork),
('92a9b284-6073-47e9-8eee-5f2195d0879a', OTD.Trail),
('96c508b9-607d-4ace-a0ba-cf5082ad6441', OTD.TransportNetwork),
('96c508b9-607d-4ace-a0ba-cf5082ad6441', OTD.Trail),
('99043ac8-9e0d-4cd6-bb1d-7e45adc22556', OTD.TransportNetwork),
('99043ac8-9e0d-4cd6-bb1d-7e45adc22556', OTD.Trail),
('9bc820a1-aea6-4619-a5dd-d375f30c90e3', OTD.TransportNetwork),
('9bc820a1-aea6-4619-a5dd-d375f30c90e3', OTD.Trail),
('a07a14eb-5b51-4484-ab0e-b8ec2bfdb2fa', OTD.TransportNetwork),
('a07a14eb-5b51-4484-ab0e-b8ec2bfdb2fa', OTD.Trail),
('a441154d-1c7b-44e9-9d37-2d06b6af6411', OTD.TransportNetwork),
('a441154d-1c7b-44e9-9d37-2d06b6af6411', OTD.Trail),
('a5b546fc-b9b1-4766-bd72-866e0edb874d', OTD.TransportNetwork),
('a5b546fc-b9b1-4766-bd72-866e0edb874d', OTD.Trail),
('a7e7242e-006f-445a-aee3-928ac6dbead4', OTD.TransportNetwork),
('a7e7242e-006f-445a-aee3-928ac6dbead4', OTD.Trail),
('a85bd417-117e-4a63-bf95-65f3ba0852a3', OTD.TransportNetwork),
('a85bd417-117e-4a63-bf95-65f3ba0852a3', OTD.Trail),
('b143d042-cd64-4b0c-9614-17ea71b454a4', OTD.TransportNetwork),
('b143d042-cd64-4b0c-9614-17ea71b454a4', OTD.Trail),
('b58d8633-0a0c-438c-b283-c4b0ae13cd77', OTD.TransportNetwork),
('b58d8633-0a0c-438c-b283-c4b0ae13cd77', OTD.Trail),
('bbd70a3e-8fe0-47e5-9624-880beeb19c59', OTD.TransportNetwork),
('bbd70a3e-8fe0-47e5-9624-880beeb19c59', OTD.Trail),
('bf627d4a-f115-41a2-82b9-d19de3cd5414', OTD.TransportNetwork),
('bf627d4a-f115-41a2-82b9-d19de3cd5414', OTD.Trail),
('c1392a5e-675e-4e1a-9e79-ed4e0c17328a', OTD.TransportNetwork),
('c1392a5e-675e-4e1a-9e79-ed4e0c17328a', OTD.Trail),
('c8ac90a3-6525-4653-88af-d897aab86704', OTD.TransportNetwork),
('c8ac90a3-6525-4653-88af-d897aab86704', OTD.Trail),
('d0b9c7e6-29e2-411c-9036-6f7d7e7a2c02', OTD.TransportNetwork),
('d0b9c7e6-29e2-411c-9036-6f7d7e7a2c02', OTD.Trail),
('d0ed874c-14ca-4166-a87d-23844fbf114c', OTD.TransportNetwork),
('d0ed874c-14ca-4166-a87d-23844fbf114c', OTD.Trail),
('d105d776-596e-4801-976a-c2b465b04f87', OTD.TransportNetwork),
('d105d776-596e-4801-976a-c2b465b04f87', OTD.Trail),
('d93c1ea6-4c2d-4df4-acf4-1c57b15e5790', OTD.TransportNetwork),
('d93c1ea6-4c2d-4df4-acf4-1c57b15e5790', OTD.Trail),
('ecb58c29-3169-4eef-8873-1483d1c5e88c', OTD.TransportNetwork),
('ecb58c29-3169-4eef-8873-1483d1c5e88c', OTD.Trail),
('f6475ec0-099f-4cc8-84f7-4f6b3aaae393', OTD.TransportNetwork),
('f6475ec0-099f-4cc8-84f7-4f6b3aaae393', OTD.Trail),
('3863c79d-1102-45dc-aecb-6d0c0d0a1696', OTD.TransportService),
('3863c79d-1102-45dc-aecb-6d0c0d0a1696', OTD.AirQuality),
('1b85ba90-b675-4831-87fd-4d0de893df18', OTD.TransportNetworkCondition),
    ('1b85ba90-b675-4831-87fd-4d0de893df18', OTD.RoutePlan),
    ('78a0cc73-b4de-4c87-84b0-c5bb7ba079c2', OTD.TravelInformation),
    ('e871da91-84ca-4703-ae14-2c79eed8aa5a', OTD.TravelInformation),
    ('e871da91-84ca-4703-ae14-2c79eed8aa5a', OTD.RealTime),
    ('e871da91-84ca-4703-ae14-2c79eed8aa5a', OTD.Timetable),
    ('e871da91-84ca-4703-ae14-2c79eed8aa5a', OTD.TravelPlan),
    ('e871da91-84ca-4703-ae14-2c79eed8aa5a', OTD.TransferNode),
    ('265a373b-0f79-49d8-aba9-65526bc74ce1', OTD.TravelInformation),
    ('265a373b-0f79-49d8-aba9-65526bc74ce1', OTD.RealTime),
    ('265a373b-0f79-49d8-aba9-65526bc74ce1', OTD.Timetable),
    ('265a373b-0f79-49d8-aba9-65526bc74ce1', OTD.TransportService),
    ('0e3f86cf-2334-41f3-8762-935e5f83d638', OTD.TravelInformation),
    ('0e3f86cf-2334-41f3-8762-935e5f83d638', OTD.Timetable),
    ('0e3f86cf-2334-41f3-8762-935e5f83d638', OTD.TransportService),
    ('3da97ae7-ae64-4d63-bc07-b7cf78d14b1f', OTD.TravelInformation),
    ('3da97ae7-ae64-4d63-bc07-b7cf78d14b1f', OTD.Timetable),
    ('3da97ae7-ae64-4d63-bc07-b7cf78d14b1f', OTD.TransportService),
    ('a7276da5-6d23-4d60-930c-3ad692ca59b1', OTD.TravelInformation),
    ('a7276da5-6d23-4d60-930c-3ad692ca59b1', OTD.Bicycle),
    ('a7276da5-6d23-4d60-930c-3ad692ca59b1', OTD.TravelPlan),
    ('16d3afdb-bb20-4593-929a-9fbd3f52ef05', OTD.Location)
]

# Generate similarities
for uuid,concept in dataset_concept_pairs:
    add_similarity_link(graph,
                        URIRef('http://78.91.98.234/dataset/{}'.format(uuid)),
                        concept)

# Store similarity graph
graph.serialize(destination='tagged-v0.9.json', format='json-ld')
```

F Appendix

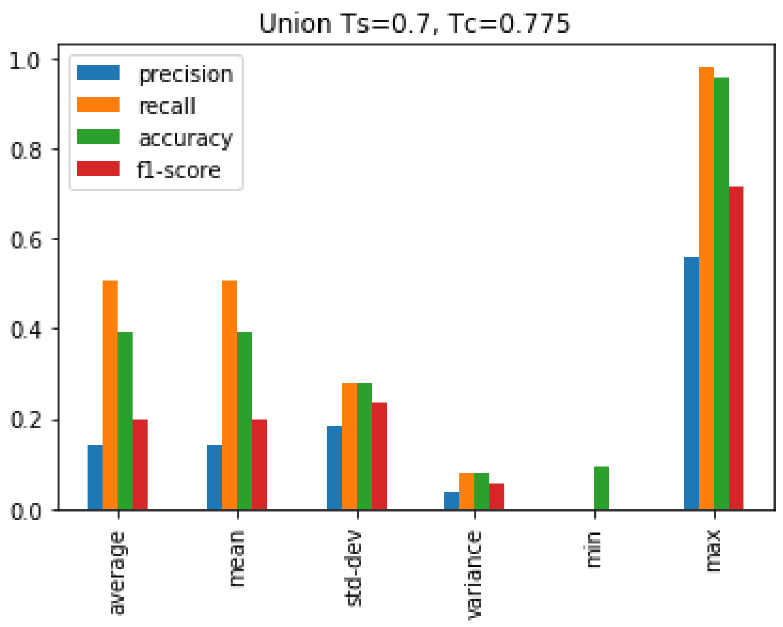
Generate Results

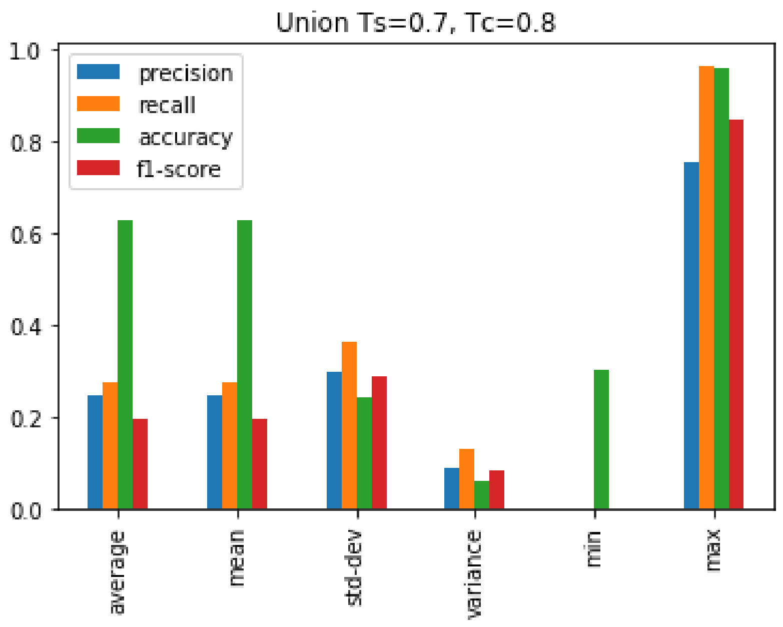
July 9, 2018

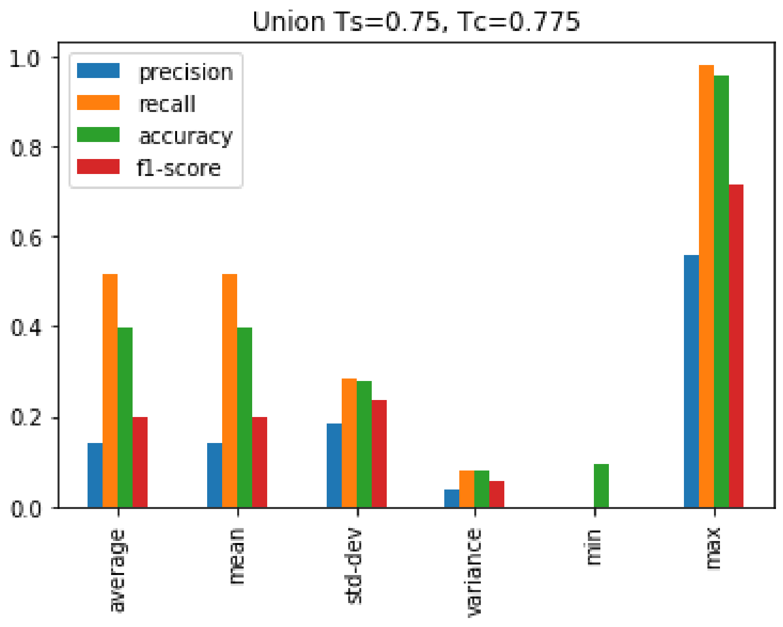
```
In [1]: %matplotlib inline
        from results.setmap import SetMap

['charging', 'station', 'tesla']


In [2]: import pandas as pd
        import matplotlib as plt
        import numpy as np
        from rdflib import Namespace
        plt.rcParams.update({'figure.max_open_warning': 0})
        OTD = Namespace('http://www.quaat.com/ontologies#')
        set_map = SetMap()

        def search_analysis(sm, query, concept, threshold, output='plot'):
            Rmanual = sm.gen_analysis(query, concept, threshold, "tagged")
            Rauto = sm.gen_analysis(query, concept, threshold, "auto")
            Runion = sm.gen_analysis(query, concept, threshold, "auto", "tagged")
            return [Rmanual, Rauto, Runion]

        def statistics(_v, label):
            total = [n[2] for n in _v]
            prec = [v[0] for v in total]
            reca = [v[1] for v in total]
            acc = [v[2] for v in total]
            f1 = [v[3] for v in total]

            df = pd.DataFrame([[np.average(prec), np.average(reca), np.average(acc), np.averag
                                [np.mean(prec), np.mean(reca), np.mean(acc), np.mean(f1)],
                                [np.std(prec), np.std(reca), np.std(acc), np.std(f1)],
                                [np.var(prec), np.var(reca), np.var(acc), np.var(f1)],
                                [np.amin(prec), np.amin(reca), np.amin(acc), np.amin(f1)],
                                [np.amax(prec), np.amax(reca), np.amax(acc), np.amax(f1)]],
                                columns=['precision', 'recall', 'accuracy', 'f1-score'],
                               index=['average', 'mean', 'std-dev', 'variance', 'min', 'max'])
            title = '{} {}'.format('Union', label)
            df.plot(kind='bar', title=title)
```

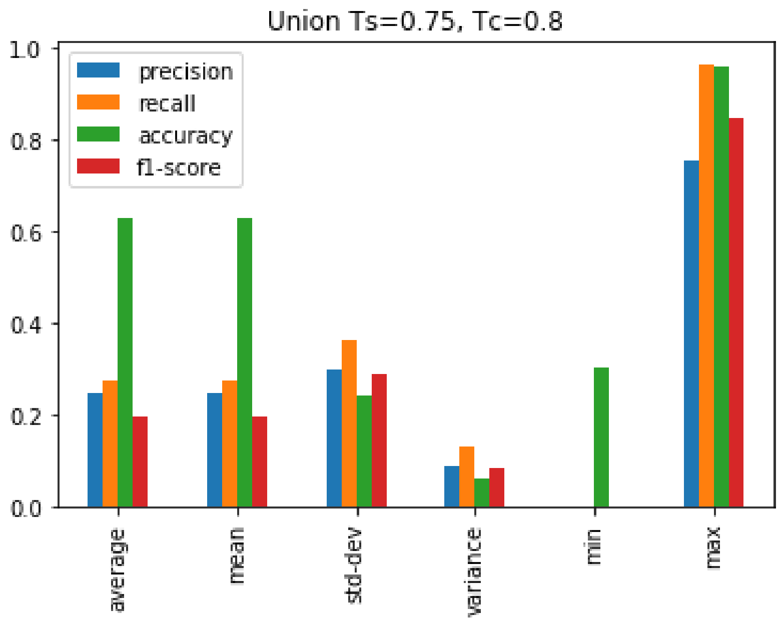
```
for search_thresh in [0.7, 0.75, 0.8]:
    set_map.build_map(search_thresh)
    for concept_thresh in [0.775, 0.80]:
        v = [
            search_analysis(set_map, "electric car charging station", OTD.ChargingStatior
            search_analysis(set_map, "information on air quality", OTD.EnvironmentInforr
            search_analysis(set_map, "location", OTD.Location, concept_thresh),
            search_analysis(set_map, "statistics", OTD.Statistics, concept_thresh),
            search_analysis(set_map, "trail", OTD.Trail, concept_thresh),
            search_analysis(set_map, "bike", OTD.Bicycle, concept_thresh),
            search_analysis(set_map, "map", OTD.Map, concept_thresh)
            ]
        statistics(v, 'Ts={}, Tc={}'.format(str(search_thresh), str(concept_thresh)))
```

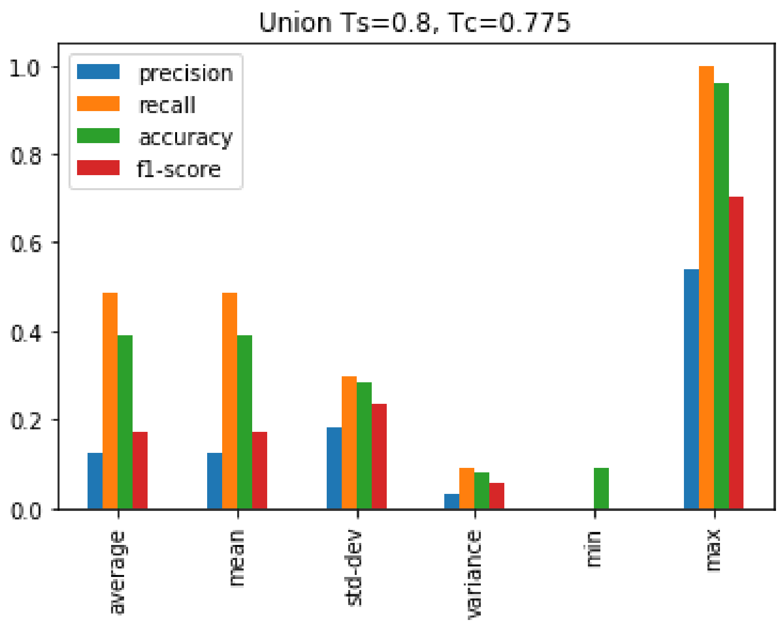
```
Indexing...
Ready
```

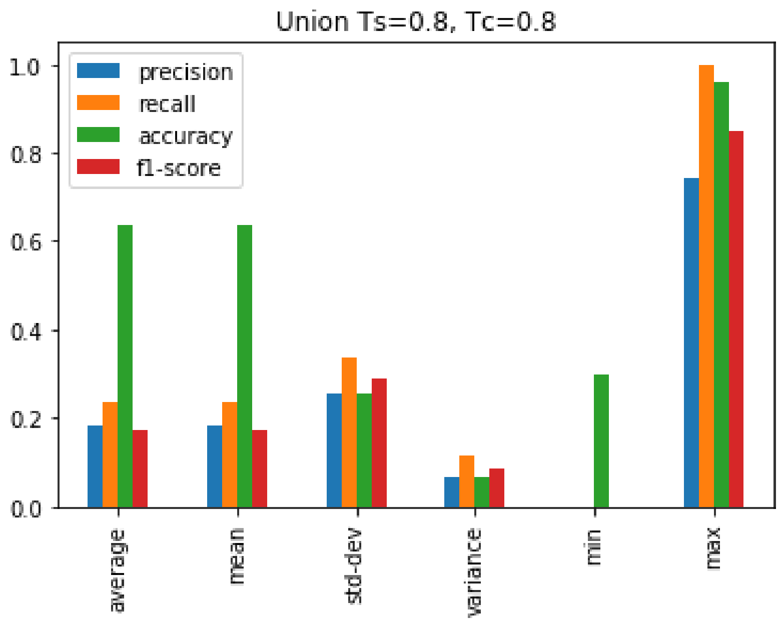



Union Ts=0.7, Tc=0.8
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
accuracy
f1-score
average
mean
std-dev
variance
min
max

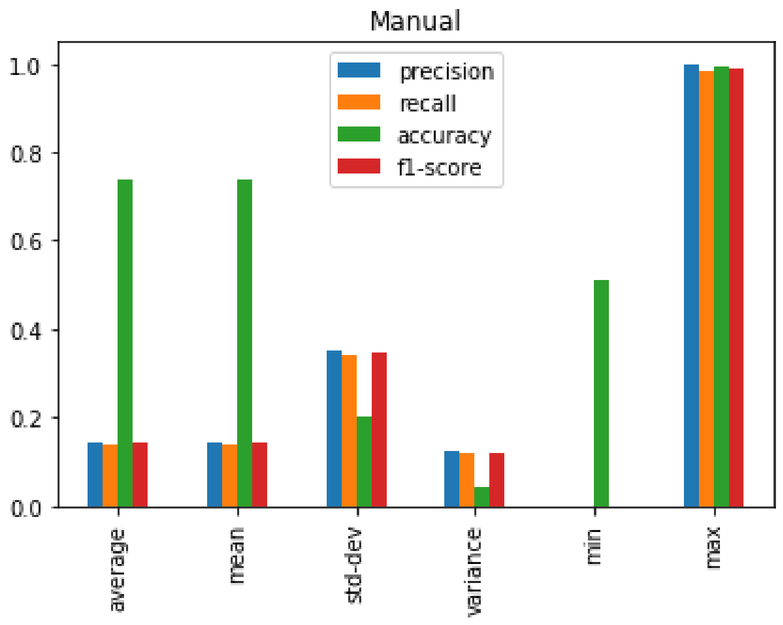

Union Ts=0.75, Tc=0.775
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
accuracy
f1-score
average
mean
std-dev
variance
min
max

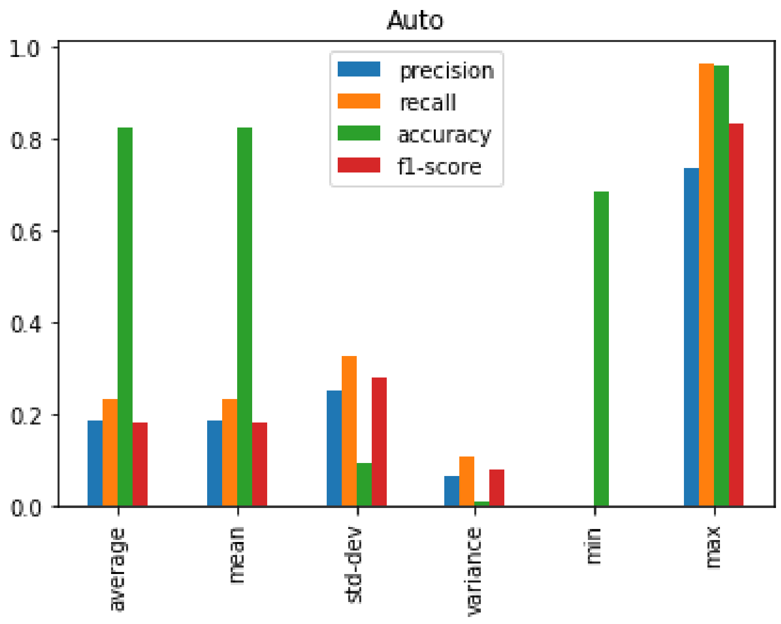

Union Ts=0.75, Tc=0.8
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
accuracy
f1-score
average
mean
std-dev
variance
min
max

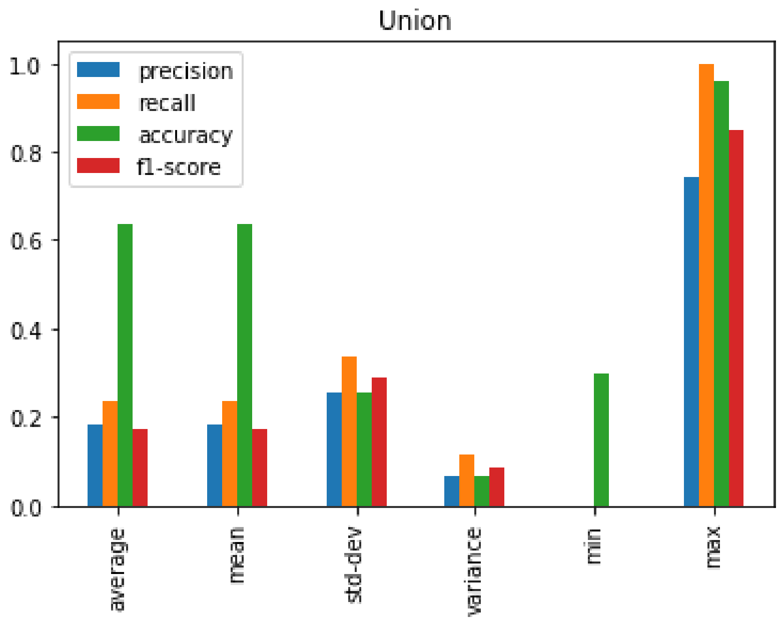

Union Ts=0.8, Tc=0.775
precision
recall
accuracy
f1-score
1.0
0.8
0.6
0.4
0.2
0.0
average
mean
std-dev
variance
min
max


Union Ts=0.8, Tc=0.8
precision
recall
accuracy
f1-score
1.0
0.8
0.6
0.4
0.2
0.0
average
mean
std-dev
variance
min
max

```
In [3]: import numpy as np
        for i,t in enumerate(['Manual','Auto','Union']):
            total = [n[i] for n in v]

            prec = [v[0] for v in total]
            reca = [v[1] for v in total]
            acc = [v[2] for v in total]
            f1 = [v[3] for v in total]

            df = pd.DataFrame([[np.average(prec), np.average(reca), np.average(acc), np.averag
                                [np.mean(prec), np.mean(reca), np.mean(acc), np.mean(f1)],
                                [np.std(prec), np.std(reca), np.std(acc), np.std(f1)],
                                [np.var(prec), np.var(reca), np.var(acc), np.var(f1)],
                                [np.amin(prec), np.amin(reca), np.amin(acc), np.amin(f1)],
                                [np.amax(prec), np.amax(reca), np.amax(acc), np.amax(f1)]],
                                columns=['precision', 'recall', 'accuracy', 'f1-score'],
                               index=['average', 'mean', 'std-dev', 'variance', 'min', 'max'])
            df.plot(kind='bar', title=t)
```




Auto
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
accuracy
f1-score
average
mean
std-dev
variance
min
max

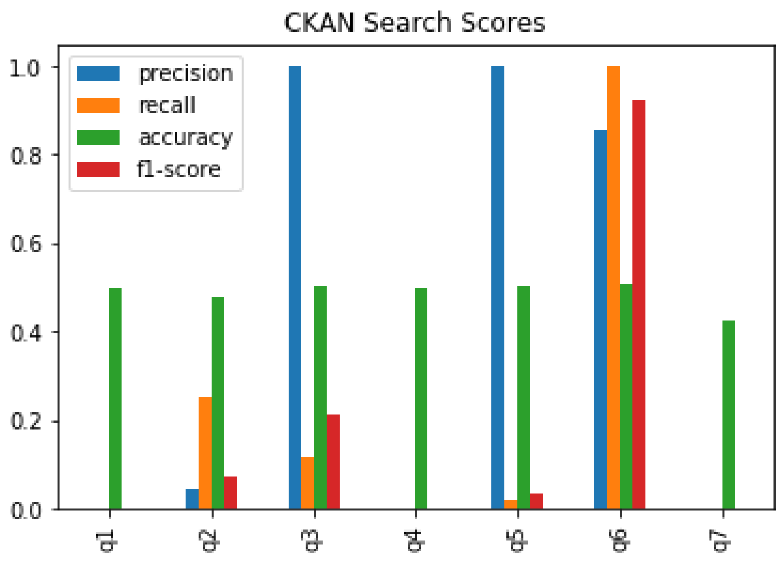

Union
1.0
0.8
0.6
0.4
0.2
0.0
precision
recall
accuracy
f1-score
average
mean
std-dev
variance
min
max

```
In [4]: def closure(val):
            value = val
            def get_next():
                nonlocal value
                value += 1
                return value

            return get_next

In [5]:  # Table showing results from search querying the CKAN server with the same seach word
         # as the semantic search engine

         # ndoc, relevant, irrelevant, relevant not found, total relevant
         #        tp    fp   fn
         a = [ 0,    0,   0,   1,   1]
         b = [ 23, 1, 22,     3,   4]
         c = [ 4,    4,   0, 30, 34]
         d = [ 3,    0,   3,   5,   5]
         e = [ 1,    1,   0, 54, 55]
         f = [ 7,    6,   1,   0,   6]
         g = [80,   0, 80,   3,   3]

         totalv = [a,b,c,d,e,f,g]
         def precision (v):
             tp = v[1]
             fp = v[2]
             fn = v[3]
             if (tp+fp) > 0:
                 return tp / (tp+fp)
             return 0.0

         def recall (v):
             tp = v[1]
             fp = v[2]
             fn = v[3]
             if (tp+fn) > 0:
                 return tp/(tp+fn)
             return 0.0

         def accuracy(v):
             tp = v[1]
             fp = v[2]
             fn = v[3]
             tn = 220-tp
             if (tp+tn+fp+fn) > 0:
```

```
        return (tp+tn)/(tp+tn+fp+tn)

def f1_score (v):
    p = precision(v)
    r = recall(v)
    if (p+r) > 0:
        return 2.0*(p*r)/(p+r)
    return 0.0

tot = []
for v in totalv:
    tot.append([precision(v), recall(v), accuracy(v), f1_score(v)])
print (tot)

df = pd.DataFrame(tot, columns=['precision', 'recall', 'accuracy', 'f1-score'],
                  index=['q1','q2', 'q3', 'q4', 'q5', 'q6', 'q7'])
df.plot(kind='bar', title='CKAN Search Scores')
df
```

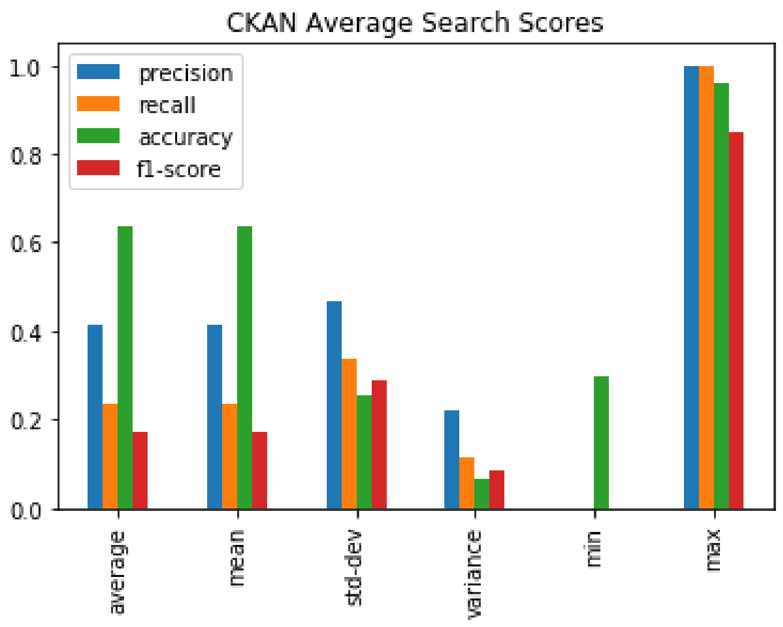
```
[[0.0, 0.0, 0.5, 0.0], [0.043478260869565216, 0.25, 0.4772234273318872, 0.07407407407407407
```

```
Out[5]:     precision    recall  accuracy  f1-score
        q1   0.000000  0.000000  0.500000  0.000000
        q2   0.043478  0.250000  0.477223  0.074074
        q3   1.000000  0.117647  0.504587  0.210526
        q4   0.000000  0.000000  0.496614  0.000000
        q5   1.000000  0.018182  0.501139  0.035714
        q6   0.857143  1.000000  0.505747  0.923077
        q7   0.000000  0.000000  0.423077  0.000000
```



```
In [6]: prec = [v[0] for v in tot]
        reca = [v[1] for v in total]
        acc  = [v[2] for v in total]
        f1   = [v[3] for v in total]

        df = pd.DataFrame([[np.average(prec), np.average(reca), np.average(acc), np.average(f1
                            [np.mean(prec), np.mean(reca), np.mean(acc), np.mean(f1)],
                            [np.std(prec), np.std(reca), np.std(acc), np.std(f1)],
                            [np.var(prec), np.var(reca), np.var(acc), np.var(f1)],
                            [np.amin(prec), np.amin(reca), np.amin(acc), np.amin(f1)],
                            [np.amax(prec), np.amax(reca), np.amax(acc), np.amax(f1)]],
                            columns=['precision', 'recall', 'accuracy', 'f1-score'],
                           index=['average', 'mean', 'std-dev', 'variance', 'min', 'max'])
        df.plot(kind='bar', title='CKAN Average Search Scores')
        df
```

```
Out[6]:          precision    recall  accuracy  f1-score
        average   0.414374  0.234354  0.636905  0.174289
        mean      0.414374  0.234354  0.636905  0.174289
        std-dev   0.468225  0.336457  0.256459  0.288518
        variance  0.219234  0.113204  0.065771  0.083243
        min       0.000000  0.000000  0.300000  0.000000
        max       1.000000  1.000000  0.958333  0.850394
```


CKAN Average Search Scores
precision
recall
accuracy
f1-score
1.0
0.8
0.6
0.4
0.2
0.0
average
mean
std-dev
variance
min
max

G Appendix

SetMap Class
Generating a confusion matrix based on the actual and expected search results.
July 9, 2018

In [ ]:
```
# Automatic search-through
import numpy as np
from odt.database import load_ontology, load_dataset, load_similarity, load_autotagge
from odt.opendatasemanticframework import OpenDataSemanticFramework
from rdflib import URIRef
from rdflib import Namespace
from results.confusion import confusion_matrix_scores, precision, recall, true_negative

OTD = Namespace('http://www.quaat.com/ontologies#')
class SetMap():
    global OTD

    # Fetch ontology, dataset, auto classified and manually tagged similarity graph.
    def __init__(self):
        self.config = {}
        self.config['ONTOLOGY_UUID']   = '5b2ab51401d5412566cf4e94'
        self.config['DATASETS_UUID']   = '5b2968c501d5412566cf4e86'
        self.config['SIMILARITY_UUID'] = '5b2ada4d01d5412566cf4ea1'
        self.config['AUTOTAG_UUID']    = '5b2acdfe01d5412566cf4e99'

        self.uri = 'mongodb://{0}:{1}@ds119969.mlab.com:19969/ontodb'.format('<use
        print ("Indexing...")
        self.ontology_graph = load_ontology(self.uri, self.config['ONTOLOGY_UUID'])
        self.datasets_graph = load_dataset(self.uri, self.config['DATASETS_UUID'])

        # Use preindexed cache for speedup
        CcsId    = "5b2adb9e01d5414513ea9802"
        AutoID   = "5b2adb9c01d5414513ea9800"
        ManualID = "5b2adb3401d5414513ea97fe"

        self.ontology = OpenDataSemanticFramework(self.ontology_graph, self.datasets_
        self.ontology.load_ccs(self.uri, CcsId)
        self.ontology.load_similarity_graph("tagged", self.uri, ManualID)
        self.ontology.load_similarity_graph("auto", self.uri, AutoID)
        self.set_map = {}
        self.all_datasets = set()
        print ("Ready")
```

```
def build_set_map(self, uri, group, threshold=0.7):
    self.all_datasets.add(uri)
    related = self.ontology.related_datasets(group, threshold)
    for r in related:
        if not r in self.set_map:
            self.set_map[r] = set()
        self.set_map[r].add(uri)

def build_map(self, threshold=0.7):
    self.set_map = {}
    self.all_datasets = set()

    self.build_set_map('http://78.91.98.234/dataset/caefad21-41cc-41e6-9f59-e05048
    self.build_set_map('http://78.91.98.234/dataset/88d19fa6-3c12-4838-b58e-38d28
    self.build_set_map('http://78.91.98.234/dataset/d473bc01-8d6e-4771-a946-e19d3
    self.build_set_map('http://78.91.98.234/dataset/1f64a769-9c10-4cc7-9db9-60ac7
    self.build_set_map('http://78.91.98.234/dataset/3e661b0d-d325-4ca4-b536-972b6
    self.build_set_map('http://78.91.98.234/dataset/9f6d9caa-df3a-4515-8ba3-025c07
    self.build_set_map('http://78.91.98.234/dataset/055880c9-cb7e-4919-ab9f-e6d6e
    self.build_set_map('http://78.91.98.234/dataset/0612abbf-1b35-422c-971c-869c6
    self.build_set_map('http://78.91.98.234/dataset/0827894d-5db2-443b-b8fb-e58e6
    self.build_set_map('http://78.91.98.234/dataset/12bf3448-ec94-4188-8a1f-c8925
    self.build_set_map('http://78.91.98.234/dataset/1a7f9e36-0e6f-41e6-9608-4f184e
    self.build_set_map('http://78.91.98.234/dataset/4298b3a3-d06e-40bc-9097-4229e
    self.build_set_map('http://78.91.98.234/dataset/4fbaaeb9-b6bb-4862-b8cf-aee476
    self.build_set_map('http://78.91.98.234/dataset/64a3f8ff-5ce8-40f7-bf9d-33e335d
    self.build_set_map('http://78.91.98.234/dataset/6adb0d93-6cb3-4e4b-861f-489f3
    self.build_set_map('http://78.91.98.234/dataset/6e24e79e-3bda-4a65-87fb-8123f
    self.build_set_map('http://78.91.98.234/dataset/85cd5905-a6a0-422b-ae8c-776b9
    self.build_set_map('http://78.91.98.234/dataset/8966a9f8-c4aa-4c83-ae1c-b9de2
    self.build_set_map('http://78.91.98.234/dataset/89cf5416-caf8-46d7-9d2d-7bf57e
    self.build_set_map('http://78.91.98.234/dataset/8f8ac030-0d03-46e2-8eb7-844ee
    self.build_set_map('http://78.91.98.234/dataset/94016332-2c3f-4c5d-9798-45829
    self.build_set_map('http://78.91.98.234/dataset/9c18dcad-313d-4bb0-bec7-358c4
    self.build_set_map('http://78.91.98.234/dataset/a68c8302-5192-4a34-9134-9850
    self.build_set_map('http://78.91.98.234/dataset/a8669394-1dc6-4258-be23-09f67
    self.build_set_map('http://78.91.98.234/dataset/a8cb46ae-7441-4a87-b52e-53516
    self.build_set_map('http://78.91.98.234/dataset/ad9335f3-a405-4811-86a2-e7cb3
    self.build_set_map('http://78.91.98.234/dataset/b6feae31-74cd-4ada-bc44-0a372
    self.build_set_map('http://78.91.98.234/dataset/c028cc71-066c-42d6-8e66-31752
    self.build_set_map('http://78.91.98.234/dataset/d1a3a50b-0566-48c1-acc0-15049
    self.build_set_map('http://78.91.98.234/dataset/de82c50b-52c2-4266-9765-386a
    self.build_set_map('http://78.91.98.234/dataset/dfb9b81c-d9a2-4542-8f63-7584a
    self.build_set_map('http://78.91.98.234/dataset/eca0854c-32dd-4692-8caa-dbdbd
    self.build_set_map('http://78.91.98.234/dataset/ee2f48ab-5b04-4c80-ad3f-0c6cfb
    self.build_set_map('http://78.91.98.234/dataset/f5b758cf-9a21-419a-a311-e6ecee
    self.build_set_map('http://78.91.98.234/dataset/36ceda99-bbc3-4909-bc52-b05a6
self.build_set_map('http://78.91.98.234/dataset/36ceda99-bbc3-4909-bc52-b05a6
self.build_set_map('http://78.91.98.234/dataset/52d0d63e-d26c-43b9-91e2-13b3b
self.build_set_map('http://78.91.98.234/dataset/52d0d63e-d26c-43b9-91e2-13b3b
self.build_set_map('http://78.91.98.234/dataset/95797234-3d9b-415c-a7e2-c4660
self.build_set_map('http://78.91.98.234/dataset/95797234-3d9b-415c-a7e2-c4660
self.build_set_map('http://78.91.98.234/dataset/a487ff09-435a-44bf-8c35-36c60b
self.build_set_map('http://78.91.98.234/dataset/a487ff09-435a-44bf-8c35-36c60b
self.build_set_map('http://78.91.98.234/dataset/23fef01e-c729-43b2-8fb3-8e127f
self.build_set_map('http://78.91.98.234/dataset/0f0e037e-b5e8-453f-97ca-8ae9be
self.build_set_map('http://78.91.98.234/dataset/0f0e037e-b5e8-453f-97ca-8ae9be
self.build_set_map('http://78.91.98.234/dataset/0f0e037e-b5e8-453f-97ca-8ae9be
self.build_set_map('http://78.91.98.234/dataset/bf374070-430b-4d30-8818-ff5ca7
self.build_set_map('http://78.91.98.234/dataset/1510b486-ef6a-4c03-a858-b19a3
self.build_set_map('http://78.91.98.234/dataset/4698e844-2cc0-429a-958e-08539
self.build_set_map('http://78.91.98.234/dataset/e97fd128-4db5-4b0a-ab36-658c4
self.build_set_map('http://78.91.98.234/dataset/f2c97ee0-6cad-4833-92fb-4a21d8
self.build_set_map('http://78.91.98.234/dataset/8feb944e-b50d-4536-a61a-cbdcf5
self.build_set_map('http://78.91.98.234/dataset/8feb944e-b50d-4536-a61a-cbdcf5
self.build_set_map('http://78.91.98.234/dataset/ab60e637-ef33-44ea-b1d4-0bb0fe
self.build_set_map('http://78.91.98.234/dataset/ab60e637-ef33-44ea-b1d4-0bb0fe
self.build_set_map('http://78.91.98.234/dataset/027b3371-5d84-4ac4-8dda-bba83
self.build_set_map('http://78.91.98.234/dataset/027b3371-5d84-4ac4-8dda-bba83
self.build_set_map('http://78.91.98.234/dataset/90cef5d5-601e-4412-87e4-3e9e8
self.build_set_map('http://78.91.98.234/dataset/90cef5d5-601e-4412-87e4-3e9e8
self.build_set_map('http://78.91.98.234/dataset/42fb5664-7801-4787-9d93-7b0c8
self.build_set_map('http://78.91.98.234/dataset/42fb5664-7801-4787-9d93-7b0c8
self.build_set_map('http://78.91.98.234/dataset/9e449b4a-a7c9-4d47-9d5e-94128
self.build_set_map('http://78.91.98.234/dataset/9e449b4a-a7c9-4d47-9d5e-94128
self.build_set_map('http://78.91.98.234/dataset/52546226-9fc9-4020-bd21-bd869
self.build_set_map('http://78.91.98.234/dataset/52546226-9fc9-4020-bd21-bd869
self.build_set_map('http://78.91.98.234/dataset/aad23951-b6eb-489e-8bfe-59949
self.build_set_map('http://78.91.98.234/dataset/aad23951-b6eb-489e-8bfe-59949
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/8c920c9b-77ad-4f64-8253-08507
self.build_set_map('http://78.91.98.234/dataset/fb3cd5ed-c920-4369-aa84-e4a0a
self.build_set_map('http://78.91.98.234/dataset/fb3cd5ed-c920-4369-aa84-e4a0a
self.build_set_map('http://78.91.98.234/dataset/8f3b2912-a296-42f9-bd3c-afeea8
self.build_set_map('http://78.91.98.234/dataset/482b30a8-1b35-4ed0-85ec-cc523
self.build_set_map('http://78.91.98.234/dataset/482b30a8-1b35-4ed0-85ec-cc523
self.build_set_map('http://78.91.98.234/dataset/08c25788-f90a-452c-bde0-1511c
self.build_set_map('http://78.91.98.234/dataset/08c25788-f90a-452c-bde0-1511c
self.build_set_map('http://78.91.98.234/dataset/0a4c57ac-386e-4a9b-ad09-c3d17
self.build_set_map('http://78.91.98.234/dataset/0a4c57ac-386e-4a9b-ad09-c3d17
self.build_set_map('http://78.91.98.234/dataset/0cd2de3b-e76f-49d8-b8cc-4c101
self.build_set_map('http://78.91.98.234/dataset/0cd2de3b-e76f-49d8-b8cc-4c101
self.build_set_map('http://78.91.98.234/dataset/0e4afab3-e679-4d3d-8052-0f793
self.build_set_map('http://78.91.98.234/dataset/0e4afab3-e679-4d3d-8052-0f793
self.build_set_map('http://78.91.98.234/dataset/1284a379-64f8-4066-bf2c-155a7
self.build_set_map('http://78.91.98.234/dataset/1284a379-64f8-4066-bf2c-155a7
self.build_set_map('http://78.91.98.234/dataset/1335971d-58ad-4a95-9b60-a122
self.build_set_map('http://78.91.98.234/dataset/1335971d-58ad-4a95-9b60-a122
self.build_set_map('http://78.91.98.234/dataset/14027ce4-09de-4a1c-b2bc-61f5a
self.build_set_map('http://78.91.98.234/dataset/14027ce4-09de-4a1c-b2bc-61f5a
self.build_set_map('http://78.91.98.234/dataset/1a3f7855-45f4-473c-89af-7fda8e
self.build_set_map('http://78.91.98.234/dataset/1a3f7855-45f4-473c-89af-7fda8e
self.build_set_map('http://78.91.98.234/dataset/221b490a-b9cc-4430-9bf1-ad59c
self.build_set_map('http://78.91.98.234/dataset/221b490a-b9cc-4430-9bf1-ad59c
self.build_set_map('http://78.91.98.234/dataset/22a016c9-c87e-4c14-b955-0b8f8
self.build_set_map('http://78.91.98.234/dataset/22a016c9-c87e-4c14-b955-0b8f8
self.build_set_map('http://78.91.98.234/dataset/25a7128a-1daf-485a-967e-dc208
self.build_set_map('http://78.91.98.234/dataset/25a7128a-1daf-485a-967e-dc208
self.build_set_map('http://78.91.98.234/dataset/29078456-a2e4-4659-bd37-b2c0
self.build_set_map('http://78.91.98.234/dataset/29078456-a2e4-4659-bd37-b2c0
self.build_set_map('http://78.91.98.234/dataset/2adf0936-b181-4cd5-82f8-fff6802
self.build_set_map('http://78.91.98.234/dataset/2adf0936-b181-4cd5-82f8-fff6802
self.build_set_map('http://78.91.98.234/dataset/2bfa7959-e9bc-4f4c-b1fa-46babd
self.build_set_map('http://78.91.98.234/dataset/2bfa7959-e9bc-4f4c-b1fa-46babd
self.build_set_map('http://78.91.98.234/dataset/2e78cdcc-897f-42d7-bd78-a09e6
self.build_set_map('http://78.91.98.234/dataset/2e78cdcc-897f-42d7-bd78-a09e6
self.build_set_map('http://78.91.98.234/dataset/3433bcbe-e8cc-465f-bee4-95265
self.build_set_map('http://78.91.98.234/dataset/3433bcbe-e8cc-465f-bee4-95265
self.build_set_map('http://78.91.98.234/dataset/42e225d3-4bf0-45ca-b579-500c6
self.build_set_map('http://78.91.98.234/dataset/42e225d3-4bf0-45ca-b579-500c6
self.build_set_map('http://78.91.98.234/dataset/48b50420-74c4-4c0c-9b60-20f20
self.build_set_map('http://78.91.98.234/dataset/48b50420-74c4-4c0c-9b60-20f20
self.build_set_map('http://78.91.98.234/dataset/49f3ebf9-40a8-4dd8-bd98-e7a0c
self.build_set_map('http://78.91.98.234/dataset/49f3ebf9-40a8-4dd8-bd98-e7a0c
self.build_set_map('http://78.91.98.234/dataset/4a8324d4-f531-41bc-8964-2dc99
self.build_set_map('http://78.91.98.234/dataset/4a8324d4-f531-41bc-8964-2dc99
self.build_set_map('http://78.91.98.234/dataset/4ae81d47-ca9d-4dd8-a65c-47b5e
self.build_set_map('http://78.91.98.234/dataset/4ae81d47-ca9d-4dd8-a65c-47b5e
self.build_set_map('http://78.91.98.234/dataset/4c13d555-fdfe-48b1-bfe6-db3c96
self.build_set_map('http://78.91.98.234/dataset/4c13d555-fdfe-48b1-bfe6-db3c96
self.build_set_map('http://78.91.98.234/dataset/4cee70c8-2fdc-4d25-a8e7-3daa2
self.build_set_map('http://78.91.98.234/dataset/4cee70c8-2fdc-4d25-a8e7-3daa2
self.build_set_map('http://78.91.98.234/dataset/51f6952b-069d-4af8-9a43-425ec
self.build_set_map('http://78.91.98.234/dataset/51f6952b-069d-4af8-9a43-425ec
self.build_set_map('http://78.91.98.234/dataset/53a3c05b-973a-4f8d-9ec3-c2fdde
self.build_set_map('http://78.91.98.234/dataset/53a3c05b-973a-4f8d-9ec3-c2fdde
self.build_set_map('http://78.91.98.234/dataset/584295a1-248c-4a35-b8d5-3fa2d
self.build_set_map('http://78.91.98.234/dataset/584295a1-248c-4a35-b8d5-3fa2d
self.build_set_map('http://78.91.98.234/dataset/5bb5d4d6-57db-4bb8-9c0b-e1e4
self.build_set_map('http://78.91.98.234/dataset/5bb5d4d6-57db-4bb8-9c0b-e1e43
self.build_set_map('http://78.91.98.234/dataset/5e5bab6c-0659-4f95-b65e-85fb45
self.build_set_map('http://78.91.98.234/dataset/5e5bab6c-0659-4f95-b65e-85fb45
self.build_set_map('http://78.91.98.234/dataset/5e604a17-dab3-4e6a-977d-ff4964
self.build_set_map('http://78.91.98.234/dataset/5e604a17-dab3-4e6a-977d-ff4964
self.build_set_map('http://78.91.98.234/dataset/64938d2c-859b-4370-8f58-9dce5
self.build_set_map('http://78.91.98.234/dataset/64938d2c-859b-4370-8f58-9dce5
self.build_set_map('http://78.91.98.234/dataset/650f6dc3-7d13-4472-a83a-3571a
self.build_set_map('http://78.91.98.234/dataset/650f6dc3-7d13-4472-a83a-3571a
self.build_set_map('http://78.91.98.234/dataset/6532c035-257d-4808-9883-38614
self.build_set_map('http://78.91.98.234/dataset/6532c035-257d-4808-9883-38614
self.build_set_map('http://78.91.98.234/dataset/725a493d-2942-4585-b72f-df6d1b
self.build_set_map('http://78.91.98.234/dataset/725a493d-2942-4585-b72f-df6d1b
self.build_set_map('http://78.91.98.234/dataset/92a9b284-6073-47e9-8eee-5f219
self.build_set_map('http://78.91.98.234/dataset/92a9b284-6073-47e9-8eee-5f219
self.build_set_map('http://78.91.98.234/dataset/96c508b9-607d-4ace-a0ba-cf5082
self.build_set_map('http://78.91.98.234/dataset/96c508b9-607d-4ace-a0ba-cf5082
self.build_set_map('http://78.91.98.234/dataset/99043ac8-9e0d-4cd6-bb1d-7e45a
self.build_set_map('http://78.91.98.234/dataset/99043ac8-9e0d-4cd6-bb1d-7e45a
self.build_set_map('http://78.91.98.234/dataset/9bc820a1-aea6-4619-a5dd-d375f
self.build_set_map('http://78.91.98.234/dataset/9bc820a1-aea6-4619-a5dd-d375f
self.build_set_map('http://78.91.98.234/dataset/a07a14eb-5b51-4484-ab0e-b8ec2
self.build_set_map('http://78.91.98.234/dataset/a07a14eb-5b51-4484-ab0e-b8ec2
self.build_set_map('http://78.91.98.234/dataset/a441154d-1c7b-44e9-9d37-2d06b
self.build_set_map('http://78.91.98.234/dataset/a441154d-1c7b-44e9-9d37-2d06b
self.build_set_map('http://78.91.98.234/dataset/a5b546fc-b9b1-4766-bd72-866e0
self.build_set_map('http://78.91.98.234/dataset/a5b546fc-b9b1-4766-bd72-866e0
self.build_set_map('http://78.91.98.234/dataset/a7e7242e-006f-445a-aee3-928ac0
self.build_set_map('http://78.91.98.234/dataset/a7e7242e-006f-445a-aee3-928ac0
self.build_set_map('http://78.91.98.234/dataset/a85bd417-117e-4a63-bf95-65f3ba
self.build_set_map('http://78.91.98.234/dataset/a85bd417-117e-4a63-bf95-65f3ba
self.build_set_map('http://78.91.98.234/dataset/b143d042-cd64-4b0c-9614-17ea7
self.build_set_map('http://78.91.98.234/dataset/b143d042-cd64-4b0c-9614-17ea7
self.build_set_map('http://78.91.98.234/dataset/b58d8633-0a0c-438c-b283-c4b0a
self.build_set_map('http://78.91.98.234/dataset/b58d8633-0a0c-438c-b283-c4b0a
self.build_set_map('http://78.91.98.234/dataset/bbd70a3e-8fe0-47e5-9624-880be
self.build_set_map('http://78.91.98.234/dataset/bbd70a3e-8fe0-47e5-9624-880be
self.build_set_map('http://78.91.98.234/dataset/bf627d4a-f115-41a2-82b9-d19de3
self.build_set_map('http://78.91.98.234/dataset/bf627d4a-f115-41a2-82b9-d19de3
self.build_set_map('http://78.91.98.234/dataset/c1392a5e-675e-4e1a-9e79-ed4e0
self.build_set_map('http://78.91.98.234/dataset/c1392a5e-675e-4e1a-9e79-ed4e0
self.build_set_map('http://78.91.98.234/dataset/c8ac90a3-6525-4653-88af-d897aa
self.build_set_map('http://78.91.98.234/dataset/c8ac90a3-6525-4653-88af-d897aa
self.build_set_map('http://78.91.98.234/dataset/d0b9c7e6-29e2-411c-9036-6f7d7
self.build_set_map('http://78.91.98.234/dataset/d0b9c7e6-29e2-411c-9036-6f7d7
self.build_set_map('http://78.91.98.234/dataset/d0ed874c-14ca-4166-a87d-23844
self.build_set_map('http://78.91.98.234/dataset/d0ed874c-14ca-4166-a87d-23844
self.build_set_map('http://78.91.98.234/dataset/d105d776-596e-4801-976a-c2b46
        self.build_set_map('http://78.91.98.234/dataset/d105d776-596e-4801-976a-c2b46
        self.build_set_map('http://78.91.98.234/dataset/d93c1ea6-4c2d-4df4-acf4-1c57b1
        self.build_set_map('http://78.91.98.234/dataset/d93c1ea6-4c2d-4df4-acf4-1c57b1
        self.build_set_map('http://78.91.98.234/dataset/ecb58c29-3169-4eef-8873-1483d
        self.build_set_map('http://78.91.98.234/dataset/ecb58c29-3169-4eef-8873-1483d
        self.build_set_map('http://78.91.98.234/dataset/f6475ec0-099f-4cc8-84f7-4f6b3aa
        self.build_set_map('http://78.91.98.234/dataset/f6475ec0-099f-4cc8-84f7-4f6b3aa
        self.build_set_map('http://78.91.98.234/dataset/3863c79d-1102-45dc-aecb-6d0c0
        self.build_set_map('http://78.91.98.234/dataset/3863c79d-1102-45dc-aecb-6d0c0
        self.build_set_map('http://78.91.98.234/dataset/1b85ba90-b675-4831-87fd-4d0de
        self.build_set_map('http://78.91.98.234/dataset/1b85ba90-b675-4831-87fd-4d0de
        self.build_set_map('http://78.91.98.234/dataset/78a0cc73-b4de-4c87-84b0-c5bb7
        self.build_set_map('http://78.91.98.234/dataset/e871da91-84ca-4703-ae14-2c79e
        self.build_set_map('http://78.91.98.234/dataset/e871da91-84ca-4703-ae14-2c79e
        self.build_set_map('http://78.91.98.234/dataset/e871da91-84ca-4703-ae14-2c79e
        self.build_set_map('http://78.91.98.234/dataset/e871da91-84ca-4703-ae14-2c79e
        self.build_set_map('http://78.91.98.234/dataset/e871da91-84ca-4703-ae14-2c79e
        self.build_set_map('http://78.91.98.234/dataset/265a373b-0f79-49d8-aba9-65526
        self.build_set_map('http://78.91.98.234/dataset/265a373b-0f79-49d8-aba9-65526
        self.build_set_map('http://78.91.98.234/dataset/265a373b-0f79-49d8-aba9-65526
        self.build_set_map('http://78.91.98.234/dataset/265a373b-0f79-49d8-aba9-65526
        self.build_set_map('http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f
        self.build_set_map('http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f
        self.build_set_map('http://78.91.98.234/dataset/0e3f86cf-2334-41f3-8762-935e5f
        self.build_set_map('http://78.91.98.234/dataset/3da97ae7-ae64-4d63-bc07-b7cf7
        self.build_set_map('http://78.91.98.234/dataset/3da97ae7-ae64-4d63-bc07-b7cf7
        self.build_set_map('http://78.91.98.234/dataset/3da97ae7-ae64-4d63-bc07-b7cf7
        self.build_set_map('http://78.91.98.234/dataset/a7276da5-6d23-4d60-930c-3ad69
        self.build_set_map('http://78.91.98.234/dataset/a7276da5-6d23-4d60-930c-3ad69
        self.build_set_map('http://78.91.98.234/dataset/a7276da5-6d23-4d60-930c-3ad69
        self.build_set_map('http://78.91.98.234/dataset/16d3afdb-bb20-4593-929a-9fbd3

    def gen_analysis(self, query, expected_tag, threshold, *e, **opts):
        search_res = set()
        for tag_type in e:
            res = self.ontology.search_query(query, tag_type, threshold)
            for r in res:
                search_res.add(r[1][2])
        cms = confusion_matrix_scores(search_res, self.set_map[expected_tag], self.all_d
        return [precision(cms),recall(cms),accuracy(cms),f1_score(cms)]
```