

Thorben Werner Sjøstrøm Dahl

Improved Ontology Based Semantic Search for Open Data

Master's thesis in MIT
Supervisor: Jingyue Li
June 2019

Thorben Werner Sjøstrøm Dahl

Improved Ontology-Based Semantic Search for Open Data

Master's thesis in Informatics
Supervisor: Jingyue Li
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

 **NTNU**
Norwegian University of
Science and Technology

Abstract

As governments publish more datasets, potential users need a way of finding the datasets they desire. This is not always so easy. Special domain-specific terminology may seem obvious to the dataset publishers, but is likely not so obvious to the users looking for datasets. Existing keyword-based search approaches therefore leave some room for improvement, since they require that queries mention words from the dataset metadata.

Efforts have been made to help solve this problem. Hagelien’s thesis serves as a basis for this thesis, and presented a prototype for ontology-based semantic search called DataOntoSearch. His approach is to match the user’s query with concepts in an ontology, which are then matched with datasets based on either an automatic or a manual association between datasets and concepts. By using concepts as an intermediate layer, the user’s query is more likely to match with datasets that are related without mentioning the specific words used by the user.

Other efforts include Google Dataset Search, which was introduced in September 2018. Their system works much like the regular Google search, letting you search for datasets independent of where they are published, but it exhibits the same traits as other keyword-based search engines. Another approach uses the semantic web to let the user specify a time period and geographical area by using terms known to the them, rather than specific dates and coordinates. Even then, it does not look like any approach exists using an ontology for open data search.

This thesis poses two research questions: What problems are there in DataOntoSearch, and how can we address these issues? A usability test showed that the search did not return what users search for, and looked unprofessional. The users were more impressed by Google Dataset Search.

Based on these findings and other observations about DataOntoSearch, I have developed a version 2 of the system, where it has been made **1)** ready to set up and deploy **2)** interoperable with web APIs **3)** integrated with a common dataset archival solution (CKAN) **4)** hopefully more usable and **5)** better for searching .

Specifically, while version 1 had an F1 score of 6% and a MAP measure of 3%, version 2 has an F1 score of 35% and a MAP measure of 32% using the manually created dataset-concept associations. Though Google Dataset Search was included in the evaluation and scored a lower than DataOntoSearch version 2, it suffered from a bug that day which made it not perform as well as it has performed other days.

Overall, the ontology-based semantic search approach is promising, and may very well help solve the problem of making open datasets discoverable. With the contributions of this thesis, DataOntoSearch is better poised to serve its purpose “out in the wild” than ever before, and has a greater chance of being adapted and make a difference for users wishing to find a dataset to solve their problems.

Sammendrag

Etterhvert som mengden publiserte datasett har økt, har det blitt viktigere og viktigere med verktøy som hjelper brukerne med å finne fram i havet av datasett. Typiske nøkkelord-baserte søk gjør ikke så mye nytte for seg, siden domene-spesifikke faguttrykk ofte kan bli brukt av de som publiserer datasett. Når vanlige brukere skal forsøke å finne datasett, er det ikke sikkert de har kjennskap til de samme faguttrykkene, så deres spørringer returnerer ikke nødvendigvis de relevante datasettene.

Flere forsøk har blitt gjort for å løse dette problemet. Masteroppgaven til Hagelien fungerer som et grunnlag for denne masteroppgaven, og presenterte en prototype for ontologi-basert semantisk søk kalt DataOntoSearch. Hans system knytter brukerens spørring opp mot konsepter i ontologien, som igjen sammenliknes med datasett basert på enten manuelle eller automatiske koblinger mellom datasett og konsept. Ved å bruke konsepter som et mellomlag mellom spørringen og datasett, kan brukerens spørring returnere datasett som er relatert til spørringen uten at de nødvendigvis bruker de samme ordene som spørringen.

Andre systemer inkluderer Google Dataset Search, som ble introdusert i september 2018. Deres system virker på samme måte som vanlig Google-søk, ved at du kan søke etter datasett uavhengig av hvor de er publisert. Det virker dog til å ha de samme problemene som andre nøkkelord-baserte søk. En annen tilnærming lar brukerne filtrere datasett på tidsperioder og geografiske områder ved å bruke kjente navn, i stedet for å kreve spesifikke datoer og koordinater. Til tross for dette later det ikke til å være noen andre tilnærminger som bruker en ontologi for søk i åpne data.

Denne oppgaven stiller to forskningsspørsmål: Hvilke problemer har DataOntoSearch, og hvordan kan de løses? Brukbarhetstester viste at systemet ikke returnerte det brukerne så etter og så uprofesjonelt ut. De likte derimot Google Dataset Search.

Basert på dette og andre observasjoner har jeg utviklet en versjon 2 av systemet. Systemet har blitt **1**) klargjort for bruk og produksjonssetting **2**) mulig å bruke med andre systemer gjennom vev-API **3**) integrert med en mye brukt løsning for publisering av datasett (CKAN) **4**) forhåpentligvis mer brukbart, og **5**) bedre til søking.

Spesifikt så har versjon 2 en F1-skår på 35% og en målt MAP på 32%, sammenliknet med versjon 1 sin F1-skår på 6% og målt MAP på 3% med manuelt genererte datasett-konsept koblinger. Selv om Google Dataset Search var med i evalueringen og skåret lavere enn DataOntoSearch versjon 2, sleit det med en programfeil den dagen som gjorde at det ikke presterte så bra som det ellers har gjort.

Når alt kommer til stykket har ontologi-basert semantisk søk et potensiale, og kan faktisk hjelpe til å gjøre åpne datasett synlige. Med bidragene fra denne masteroppgaven står DataOntoSearch bedre rustet enn noensinne til å gjøre sitt oppdrag “i det fri,” og har en bedre sjanse til å bli adaptert og gjøre en forskjell for brukere som ønsker å finne fram til datasett som kan løse problemene deres.

Preface

For the last year, I have spent my days researching and writing this master thesis. It has been a long ride, filled both with moments of frustration and of joy. It has also been an exercise in tenacity and self discipline, and a constant balancing act between school, voluntary activities and free time at home. I am left with a greater appreciation of my own limits and a desire to start a normal work-life, where I am compensated for the hours I put in and not thinking of any free time as time I could have spent improving the thesis.

This master thesis is a part of the Open Transport Data project¹, which is a project funded by the Research Council of Norway. The project partners are:

- Norwegian Public Road Administration
- Norwegian Coastal Administration
- Municipality of Oslo
- ITS Norway
- URBALURBA
- SINTEF Digital

The work has been done under the skillful supervision of Dr. Shanshan Jiang (SINTEF) and associate professor Jingyue Li (NTNU). I would like to thank them for their time and knowledge, and SINTEF for their collaboration with NTNU for this thesis. Additionally, I would like to thank Dr. Shanshan Jiang and Marit Natvig at SINTEF for their work in updating the ontology and creating a new evaluation framework based on New York's open data.

I would also like to thank Thomas Hagelien (SINTEF) for dedicating time to explaining aspects of the system and assisting in getting it to run on my computer.

Of course, I wouldn't have made it this far without a little help from my friends. I would like to thank my friends at Studentmediene i Trondheim, and give a special thank you to those of my friends who volunteered as test users for the usability tests.

¹<https://opentransportdata.wordpress.com/>

Contents

Abstract	i
Sammendrag	ii
Preface	iii
Contents	v
List of Tables	ix
List of Figures	xi
Glossary	xii
1 Introduction	1
2 Background	3
2.1 Open data and semantic technologies	3
2.1.1 Open Data	3
2.1.2 Dataset archival solutions	4
2.1.3 Linked Data and Semantic Technologies	5
2.1.4 Taxonomy of semantic search engines	7
2.1.5 Related Natural Language Processing technologies	8
2.2 Existing search function in CKAN	9
2.3 A previous project on improving the CKAN search	11
2.4 The master thesis which improved CKAN search further	12
2.4.1 Ontology development	12
2.4.2 Semantic search	13
2.4.3 Evaluation	14

3	Related Work	17
3.1	Semantic based search	17
3.1.1	Google Dataset Search	17
3.1.2	Spatio-Temporal Search	18
3.1.3	OntRank	19
3.1.4	Dataset search using semi-structured query patterns	20
3.1.5	Ontology-based query improvement	21
3.1.6	Other semantic search engines	21
3.2	Evaluation approaches of semantic search	22
3.3	System-oriented Evaluation	23
3.3.1	Precision and recall	23
3.3.2	Precision at k	24
3.3.3	R-Precision	25
3.3.4	Mean Average Precision	25
3.3.5	Receiver Operating Characteristics (ROC)	25
3.3.6	Normalized Discounted Cumulative Gain	25
3.3.7	Best Practice	26
3.3.8	Crowdsourcing relevance assessment	27
3.4	User-oriented Evaluation	27
3.4.1	Defining Usability	27
3.4.2	Evaluating usability	28
3.4.3	Evaluating Usability of Semantic Search Interfaces: An Example Study	32
4	Overall design	35
4.1	Motivation	35
4.2	Research Questions	38
4.3	Overview	38
5	Investigation and results of RQ1	41
5.1	Methodology	41
5.1.1	Goals	41
5.1.2	Test methodology	42
5.1.3	Collection and analysis of data	43
5.1.4	Test details	44
5.2	Implementation	48
5.2.1	Getting a runnable system	48
5.2.2	Finding common datasets	50
5.2.3	Updated ontology	51
5.2.4	User interface improvements	52
5.3	Results	55
5.3.1	Changes made from pilot study	55
5.3.2	User rating of systems	56
5.3.3	User satisfaction	57
5.3.4	Problems and themes in user comments	57
5.3.5	User performance	58

6	Implementation and results of RQ2	59
6.1	Methodology	59
6.1.1	Goals	59
6.1.2	Measures	60
6.1.3	Queries and relevance assessments	60
6.1.4	Collection and analysis of data	63
6.2	Implementation	66
6.2.1	CKAN Extension	66
6.2.2	API	71
6.2.3	Improving how the query is associated with concepts	73
6.2.4	Increasing the ontology's importance	76
6.2.5	New threshold variables	77
6.2.6	Autotagging	78
6.2.7	Manual tagging process	79
6.2.8	Web interface	79
6.3	Results	83
6.3.1	Sensitivity of threshold variables	83
6.3.2	Search engine comparison	84
6.3.3	Performance	89
7	Discussion	99
7.1	The usability test	99
7.1.1	Limitations	102
7.2	The CKAN plugin	102
7.2.1	Limitations	103
7.3	Code quality	103
7.4	Search quality and evaluation	104
7.4.1	Limitations	105
7.5	Run-time performance	106
7.6	Comparison with some other approaches	106
7.7	Future work	108
8	Conclusion	111
	Bibliography	113
	Appendices	117
A	Material provided to test users in pre-study	119
A.1	Test description	120
A.2	Task descriptions	121
A.2.1	Internal notes about the tasks	121
A.3	Manual	122
A.3.1	Dataset publishers	122
A.3.2	Search engines	122

B	Example of code quality improvements	125
C	Documentation	129
C.1	README of DataOntoSearch	129
C.1.1	Preparing	130
C.1.2	Usage	130
C.2	API Documentation of dataset tagger	137
C.2.1	Overview	137
C.2.2	GET /api/v1/<uuid>/concept	137
C.2.3	GET /api/v1/<uuid>/tag	138
C.2.4	POST /api/v1/<uuid>/tag	139
C.2.5	DELETE /api/v1/<uuid>/tag	139
C.2.6	DELETE /api/v1/<uuid>/dataset	140
C.3	API Documentation of search webserver	141
C.3.1	Overview	141
C.3.2	GET /api/v1/search	141
C.4	README of ckanext-dataontosearch	143
C.4.1	Requirements	143
C.4.2	Installation	143
C.4.3	Config Settings	144
C.4.4	Development Installation	144
C.4.5	Future Work	145
C.4.6	Running the Tests	145
C.4.7	Releasing a New Version of ckanext-dataontosearch	146
D	Code used for search	147
D.1	The OpenDataSemanticFramework class	147
D.2	QueryExtractor	154
D.3	SemScore	155

List of Tables

5.1	SUS scores	57
5.2	Summary of measurements collected from users	58
6.1	Queries with concept-based relevance assessment	62
6.2	Queries derived from the pre-study	64
6.3	Previous descriptions of T_C and T_S	78
6.4	Effect of varying T_S	85
6.5	Effect of varying T_Q	85
6.6	Effect of varying T_C	86
6.7	Effect of varying T_C over T_Q , using F1 score	86
6.8	Effect of varying T_C over T_Q , using MAP	91
6.9	Effect of varying T_S over combinations of T_Q and T_C for automatic tagging, measured using F1 score	92
6.10	Effect of varying T_S over combinations of T_Q and T_C for manual tagging, measured using F1 score	93
6.11	Effect of varying T_S over combinations of T_Q and T_C for automatic tagging, measured in MAP	94
6.12	Effect of varying T_S over combinations of T_Q and T_C for manual tagging, measured in MAP	95
6.13	Summary of measurements between the two threshold combinations, using all queries. Google is included for comparison.	96
6.14	Precision measurements	96
6.15	Recall measurements	96
6.16	F1 score measurements	97
6.17	R-Precision measurements	97
6.18	Mean Average Precision (MAP) measurements	98
6.19	Summary of measurements for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.	98
6.20	Mean Average Precision (MAP) using task-based relevance assessment	98

6.21 The time spent on the different parts of the query process. Measured in real time.	98
----------------------------------------------------------------------------------------------------	----

List of Figures

2.1	Search example in CKAN	10
2.2	Overview over the search process of DataOntoSearch version 1	13
3.1	The four interfaces created in [26]. Upper left is the free-form NLI, upper right the somewhat guided NLI, lower left the very guided NLI and lower right the graphical SPARQL editor. The original screenshots are property of the paper’s authors.	33
5.1	The revised web search interface, without concepts being shown.	53
5.2	The revised web search interface, this time with concepts.	54
6.1	A list of concepts are shown for this dataset.	68
6.2	The user can add, change or remove concepts for the selected dataset. . .	68
6.3	The semantic search page lists matching concepts on the left, and the search results in the main part of the page.	69
6.4	A link to view associated concepts is shown when viewing a dataset in CKAN.	69
6.5	A link to edit associated concepts is shown when editing a dataset in CKAN. .	70
6.6	When using the CKAN search, the user can click a link to perform the same search using DataOntoSearch’s semantic search.	70
6.7	Example of query parameters in API request	73
6.8	Example of result from API request	81
6.9	Example of how the generated CSV file can be used for manual tagging. .	82
A.1	An excerpt from the ontology hierarchy.	123

Glossary

Symbol	Definition
API	Application Programming Interface. A way to make functionality available to other computer programs
CKAN	Open source software for storing, organizing and publishing datasets. Commonly used by governments, especially European ones
CLI	Command Line Interface. A way of interacting with a computer program by running it as a program on the command line
DCAT	Data Catalog Vocabulary. RDF predicates useful for describing datasets and their metadata
DIFI	Direktoratet for forvaltning og IKT (Agency for Public Management and eGovernment). The Norwegian government agency responsible for helping local governments and the central government make their datasets available publicly. They host a federated dataset catalog for this purpose
HTTP	Hypertext Transfer Protocol. The protocol used for requesting and receiving web pages.
I/O	Input/Output. Usually refers to reading input from and giving feedback to the user, reading files from or writing files to the hard drive, or communicating with other programs and computers.
Ontology	A way of organizing knowledge in a topology, i.e. a hierarchical structure where sub-concepts are specializations of their parents
OTD	Open Transport Data. Research project
OWL	Web Ontology Language. A further extension of RDFS, allowing for more inferences
NL	Natural Language. The type of writing you use when writing to another human, as opposed to the more formal writing often required when interacting with computers
NLI	Natural Language Interface. A user interface which accepts natural language from the user
NLP	Natural Language Processing. Technologies for taking natural language and transforming it into something the computer can work with

NTNU	Norwegian University of Science and Technology, located in Trondheim, Gjøvik and Ålesund
RDF	Resource Definition Language. Way of organizing knowledge, using triples of (subject, predicate, object)
RDFS	RDF Schema. Adds type functionality to RDF, letting you assign different types to different entities, and also infer types by using predicates and restrictions on what types they can apply to
REST	Representational State Transfer. Describes a way of designing an API so that it is simple to use and benefits from the nature of the web
Semantic Web	The idea of extending the World Wide Web with information readable by computers, as opposed to the unstructured text typically found on websites. Just like the regular web, one site's semantic data can refer to information found other places, essentially creating one big, distributed knowledge base
SINTEF	Norwegian research organization with a focus on technology, cooperation with NTNU and commercialization of research results
SKOS	Set of RDF predicates useful for defining an ontology, where concepts can relate to one another in a number of different ways
Socrata	Commercial alternative to CKAN, used by e.g. New York
SPARQL	SPARQL Protocol and RDF Query Language. Technology used to retrieve triples from an RDF store, using a syntax resembling SQL
SQL	Structured Query Language. Way of querying and managing data in databases
SUS	System Usability Scale. Measures how well the user felt about using the system
UUID	Universally Unique Identifier. A very random text string which is used to identify something

Introduction

Governments around the world have been adopting the *open data* philosophy, publishing their data for anyone to use in the hopes of helping innovation and be transparent. This is most often done through content management systems for open data, such as CKAN. The benefits touted by advocates have not always come to fruition, though, with datasets not being so easy to find for the average user. Even if you are walking around with a vague idea for a startup relying on some open government data, actually figuring out where to find this data – if it even exists – can be a challenge.

Clearly, there is an opportunity here to improve the discoverability of datasets. Hagelien explored different ways of doing this, eventually creating a prototype dubbed DataOntoSearch. It is an ontology-based semantic search engine for searching among datasets. By using an ontology, users do not need to know the exact words used by the dataset publishers, since the ontology adds a layer of indirection between the query and the datasets that are retrieved. DataOntoSearch seems to perform better than the search built into CKAN, at least for queries that use words different from those used in the dataset metadata.

The experimental approach has its downsides, though, and the system setup itself cannot be easily replicated by others. In addition, the evaluation of the system may be a bit unrealistic, compared to the types of queries users would make “in the wild.” There are also plenty of opportunities for improving the system itself, whether it be the user interface and the system’s ease of use, or its algorithms for retrieving and ranking datasets. Finally, a new contender entered the ring in September, with the beta launch of Google Dataset Search.

My thesis is a continuation of Hagelien’s work, and aims to answer the following two request questions:

RQ1: What do users think of DataOntoSearch version 1? What problems are there?

RQ2: How can we address the identified problems when creating version 2?

Both research questions involve deciding on a methodology, doing some implementation work and evaluating the result. For the first research question, I must be able to

run DataOntoSearch on my own computer, and ideally gain some knowledge of its inner workings. Then, I can conduct a usability test to get a more realistic and accurate view of how the system performs, and contrast it with users' experience trying Google Dataset Search.

The output of the work for the first research question functions as input for the work on the second question. Whereas the first research question is about *what* to do, the second research question mandates a more systematic evaluation so conclusions can be drawn about the improvements made. Summarized, this thesis can be said to follow a pre-study, improvements and evaluation structure.

Once I had the system running, which required a considerable amount of effort refactoring it, the usability test showed that the version of DataOntoSearch version 1 I had access to performed very badly. Users preferred Google Dataset Search, which actually helped them achieve their tasks, as opposed to DataOntoSearch which did not return the datasets the users were looking for. Users also commented that the look of DataOntoSearch's web interface did not leave them with a good impression.

For the improvement work, I have implemented a CKAN extension integrating DataOntoSearch into the popular dataset archival system. I expect that having the system available in the same website where the datasets are hosted will increase DataOntoSearch's viability as a platform for dataset search. Through this work, generic APIs have been implemented in DataOntoSearch, so it is possible to integrate it with more platforms, should it be desired. When using DataOntoSearch through CKAN, users get to enjoy the graphical user interface of CKAN instead of the prototypical interface of DataOntoSearch, hopefully fixing the problem users had with its graphical design.

Furthermore, the algorithm for mapping the user's query to concepts in the ontology has seen some improvements, by using more of the information present in the digital dictionary used (WordNet), adding some predictability to how a query can be matched with a concept by processing both the same way, and using the ontology to a greater degree. The systematic evaluation in the end shows that the new version of DataOntoSearch provides a great improvement over version 1, especially when using manually created associations between datasets in the index and concepts in the ontology. It seemingly compares favourably to Google Dataset Search, but this cannot be concluded conclusively.

The rest of the thesis is structured like this. Chapter 2 goes into the background, including an introduction to open data, semantic technologies and earlier endeavours related to the Open Transport Data project. Other related works, like other approaches for searching open data and different evaluation methods, are covered in Chapter 3. With the knowledge prerequisites out of the way, Chapter 4 details the motivation, research questions and evaluation approach, before Chapter 5 covers the first research question and 6 covers the second. A discussion of the results found is presented in Chapter 7, before Chapter 8 concludes the thesis.

Background

This master thesis is a part of the Open Transport Data project, and is a continuation of earlier efforts to make datasets more easily discoverable to dataset users. This chapter sets the stage by introducing this backdrop.

2.1 Open data and semantic technologies

2.1.1 Open Data

There are multiple ways of approaching the question of what open data is. The Open Definition project ¹ is a project of the Open Knowledge Foundation, which seeks to clarify what exactly what “openness” means. They summarize that “Open means **anyone** can **freely access, use, modify, and share** for **any purpose** (subject, at most, to requirements that preserve provenance and openness) [19].” Thus, open data can be said to mean any data which is made available in a way that conforms to this definition of “open.” The data can for example be “bus departure times in Stavanger” or “*CO*² levels measured in Tromsø,” usually made available as spreadsheets.

Of course, open data is more than just data that happens to be open. It is also an initiative which promotes the idea of opening up data, especially for government entities which generate the data using public funds. Rob Kitchin summarizes the arguments in [27]:

- **Accountability:** The general public can investigate how well the institutions in question fulfil their purpose.
- **Informed participation:** By being better informed, the general public has a greater chance of involving themselves in politics and democracy.
- **Encouraging monitoring:** Once openly available, the data can be used to monitor the institution’s performance over time, both from within the institution and by external agencies.

¹<http://opendefinition.org/>

- **Improving public image:** By adapting open practices, you are recognized as “innovative” and foster more connections with customers and end-users.
- **Generate value:** Businesses may often profit from having access to and using data produced by the government. If such data is only available at a cost, innovative new businesses may struggle to compete with the established giants who can afford the cost.

More and more governments have adopted the open data mantra, such as the US government, the European Union and also the Norwegian government.

All open data are not created equal. Just because you can download a PDF with the information does not mean you can do something useful with it. Tim Bernes-Lee has developed a five-star scale indicating how well some data accomplishes the ideals of linked, open data [5]. Using it, one star is awarded for simply making the data available on the web under an open license, while more stars are awarded as the data is made more machine-readable (e.g. going from image to Excel to CSV) and interlinked with other open data on the web (e.g. by adapting RDF and linking to other people’s data). As a dataset is awarded more stars, you can assume it has become easier to use.

Open data is not without its problems. When the data is made openly available, there is practically no way of funding the data generation, besides the government subsidizing the process. Compared to a model where businesses pay to access the data, the government essentially takes over the costs earlier paid for by those who gained the most from the data. In fact, as Rob Kitchin summarizes in [27], there are critics who say that businesses have supported open data practices as a way of generating more profit, while publicly using arguments of transparency and accountability. Critics also argue, Kitchin continues, that adapting open data may promote neoliberalization and marketisation of public services, that advocates focus on technical and economic perspectives without considering how the data may negatively affect the poor, and that the usefulness of open data has been greatly exaggerated, due to initiatives not focusing on users of the data.

2.1.2 Dataset archival solutions

Though you could simply upload your open data to a personal website, it may be hard to discover the dataset and you often want to collect datasets from many different sources and present them in a federated catalog. For example, though each municipality in Norway publish their own datasets on their own websites, the datasets are also collected and presented in one system, at <https://data.norge.no>. For this purpose, a number of solutions are available for uploading and making datasets available to the general public.

Such solutions must be easy to use, both for dataset publishers and dataset consumers. They must provide ways of finding the dataset you want to use, and also store enough metadata so you can know how and when to use the dataset, if you want to use it at all.

CKAN² is a widely used open source software for archiving datasets. The system is organized so datasets may be assigned to an organization, and each dataset has one or more resources, representing a file, typically a spreadsheet file. There are harvesting plugins available, enabling you to have a CKAN instance which harvests datasets from

²<https://ckan.org/>

many other CKAN instances, e.g. when you want a federated catalog for all municipalities. Such harvesters may harvest from many different sources, including other CKAN instances, Socrata instances and so on. CKAN's popularity means there are many plugins available for augmenting and adding functionality. Users of CKAN include the US federated catalog³ and the European Union's federated catalog, the European Data Portal⁴.

Alternatives to CKAN include the commercial service Socrata run by Tyler Technologies⁵, an open source system ready for webhotels and with integrated CMS called DKAN⁶, and an open source system focused on archival of research data called Dataverse⁷. Users of Socrata include both the city⁸ and state⁹ of New York.

In the rest of the thesis, we will focus on CKAN, since it is widely used and can be extended to handle datasets from the other dataset archival solutions.

2.1.3 Linked Data and Semantic Technologies

Making machine-readable data openly available under a non-proprietary format is just one piece of the puzzle. To truly utilize the potential of the world wide web, you want to present data in a way that lets consumers combine and look up information from multiple sources. For example, your website can refer to the city of New York. Visitors may then go to other sources describing the same entity, like the city's Wikipedia article, to learn more about said city. Semantic technologies were introduced to achieve this vision of linked, open data, often referred to as *the semantic web*.

As explained by [3]:

The Semantic Web . . . follows different design principles, which can be summarized as follows:

1. make structured and semi-structured data available in standardized formats on the web;
2. make not just the datasets, but also the individual data-elements and their relations accessible on the web,
3. describe the intended semantics of such data in a formalism, so that this intended semantics can be processed by machines

To achieve this, a couple technologies have been developed, which embed information of increasing complexity and inference technology of increasing capability. RDF is the most basic of these technologies.

³<https://catalog.data.gov>

⁴<https://www.europeandataportal.eu/en/>

⁵<https://www.tylertech.com/products/socrata>

⁶<https://getdkan.org/>

⁷<https://dataverse.org/>

⁸<https://opendata.cityofnewyork.us/>

⁹<https://data.ny.gov/>

RDF

RDF is an abbreviation of Resource Description Framework, a framework which describes a way of organizing facts about different entities and their relationships, as well as ways of representing this information in a way that computers can understand. Specifically, RDF is a family of specifications issued by the W3C organization, with the latest specification dating February 25th 2014.

Information in RDF is organized into triples. Each triple can be viewed as a simple sentence on the form “⟨*subject*⟩ ⟨*predicate*⟩ ⟨*object*⟩.” Consider the example “⟨New York⟩ ⟨is a⟩ ⟨city⟩.” “New York” is here the *subject*, “is a” plays the role of *predicate* and “city” is the *object*. As explained by the specification, a triple tells us that “some relationship, indicated by the predicate, holds between . . . the subject and object [49]”.

The subject, predicate and object can be specified using something that looks like a URL. By using them, one website can describe New York, and another website can then refer to New York by the same URL, thereby allowing systems to learn about New York from multiple sources and know that the different sources are describing the same thing.

RDF also defines ways of storing these triples. The most basic format is the XML format, but there also exist more human-friendly formats, like Turtle.

Inference

Other technologies exist that build on top of RDF, adding the capability to generate new triples by inferring information from what is already there. For example, RDFS adds the ability to assign types to entities and say that one predicate only holds between certain types. As an example, if a predicate *isTopConcept* is defined to apply between a *Concept* and a *Scheme*, and you have the triple “Animal isTopConcept AnimalScheme,” then you can infer that *Animal* must be a *Concept*, and *AnimalScheme* must be a *Scheme*, simply based on what is known about *isTopConcept*.

OWL adds even more inference power, adopting the so-called *open world assumption*, namely that instead of assuming everything is false until it is said to be true, everything is assumed to be *unknown* instead. Using hierarchies of classes, properties and entities, more complex knowledge can be represented in OWL ontologies.

Vocabularies

In order for a computer to understand RDF properly, we need to agree on some vocabularies to use. This way, whenever the same predicate is encountered across different websites, they can be taken to mean the same thing. For example, you will need to use a standard vocabulary decided by Google to get your datasets indexed by the Google Dataset Search [37].

One of the vocabularies most useful to us is the SKOS vocabulary. “The Simple Knowledge Organization System (SKOS) is an RDF vocabulary for representing semi-formal *knowledge organization systems* (KOSs), such as thesauri, taxonomies, classification schemes and subject heading lists [47].” SKOS has the tools needed to represent an ontology as a hierarchy of concepts, and attach labels and other documentary notes to each concept.

Another useful vocabulary is DCAT, i.e. the Data Catalog Vocabulary. It essentially gives you the tools to represent a catalog consisting of datasets [48]. A DCAT plugin is available in CKAN¹⁰, letting you access the dataset metadata using DCAT and RDF. DCAT has more or less become a standard way of exchanging dataset metadata between different dataset archival solutions.

2.1.4 Taxonomy of semantic search engines

Butt, Haller and Xie set out to analyze existing techniques for retrieving data from the Semantic Web, to “find the directions that have been taken” and figure out “what are some of the promising significant directions to pursue future research” [9]. As a part of their work, they created a taxonomy of different dimensions that can be used to describe different semantic search engines. Some of the most relevant dimensions are:

- **Retrieval scope:** What is searched in, and what is searched for?
 - ONTOLOGY-RETRIEVAL TECHNIQUES search for ontologies and vocabularies and inside their contents.
 - LINKED-DATA-RETRIEVAL TECHNIQUES explore the linked data which is described using these ontologies.
 - GRAPH-RETRIEVAL TECHNIQUES are made to be used on graphs, but have been applied to searching the Semantic Web.
- **Query model:** How do you model the query? And by extension, how does the user create queries?
 - In KEYWORD SEARCH, the user writes their query as a text consisting of several keywords, which most will recognize from Google.
 - STRUCTURED QUERY SEARCH is much more precise and requires that the user conforms to a certain syntax, with one example being SPARQL. It is therefore only accessible to experts with knowledge of the query language and the underlying data.
 - FACETED BROWSING uses filtering, in a way most people will recognize from e-commerce sites where you can pick what price, what brands and what type of products you would like to browse for, except the properties (i.e. facets) you can filter will vary with the domain or search results.
 - HYPERLINK-BASED TECHNIQUES offer links you can click, which will take you to new queries, letting you navigate the data.
- **Results type:** How are the retrieved results presented to the user, conceptually?
 - DOCUMENT-CENTRIC approaches simply present the retrieved data as it was found, potentially listing different information about the same thing multiple times.

¹⁰<https://github.com/ckan/ckanext-dcat>

- ENTITY-CENTRIC approaches identify the entities that are found, and collect all information about them so each entity is listed once, with all the consolidated data.
- RELATION-CENTRIC approaches focus on the relationships between entities, and is usually achieved through structured queries and faceted browsing.
- **Data acquisition:** How is the data collection done?
 - MANUAL COLLECTION requires an administrator to make a decision on what datasets to include.
 - LINKED DATA CRAWLERS crawl the Semantic Web in much the same way web search engines crawl the World Wide Web. They vary in to what degree they crawl HTML documents with embedded RDF in addition to the Semantic Web.
- **Ranking factor:** How do you rank documents?

Approaches include query-independent factors like popularity factors similar to Google’s PageRank, how trustworthy the data is, how much information the data carries, machine learning, centrality and user feedback, and the query-dependent coverage factor, which considers how well each result covers the query.
- **Datasets:** When you evaluate the engine, what kind of data do you experiment with?
 - REAL-WORLD DATA are important due to its messy and noisy nature, which search engines will need to cope with when applied to the Semantic Web.
 - SYNTHETICALLY GENERATED DATA are often used since it may be difficult to obtain real-world data of the size needed to evaluate the engines’ scalability.
- **User interface:** What kinds of usages is the engine capable of?
 - GRAPHICAL USER INTERFACES are useful for end-users.
 - APPLICATION PROGRAMMING INTERFACES (API) are useful for integrating the search with other applications.

With the dimensions defined, Butt, Haller and Xie reviewed a number of existing systems, identifying how they aligned along the dimensions. Based on this work, they identified a number of directions which had not been explored yet, like dynamically generating facets for faceted browsing, ranking of ontologies and triples, and creating a comprehensive evaluation framework for Semantic Web data retrieval techniques [9].

2.1.5 Related Natural Language Processing technologies

“Natural language” is the language we use when communicating to one another. The collection of techniques used when a computer is asked to process this type of language is called *natural language processing*, or NLP for short. This is used e.g. when trying to make sense of a search query or extracting relevant concepts from a dataset description.

Some of the NLP techniques relevant for this thesis are described below.

Tokenization The process of splitting one long text into tokens, typically one for each word and punctuation mark.

Part-of-speech tagging Analyzing the structure of sentences and the individual words to assign a part of speech, such as noun or verb, to each token.

Chunking Creating a tree out of a sentence, essentially breaking the sentence down into sub-phrases.

Lemmatization Standardizing each token so different forms of the same words are changed to a standard form, e.g. ensuring both “writes” and “wrote” are changed to “write.”

Stop word removal Removing the most common words that cannot be used to differentiate between different texts, like “a,” “the” and “is.” Removing them can often yield great performance benefits for little loss since they are so common.

WordNet

The techniques described above are not enough to gauge the “distance” in meaning between two words. For example, you intuitively know that the word “car” is more closely related to “bicycle” than to the word “flower,” but the computer does not know this. This kind of comparison would be useful to use when trying to see what concepts in an ontology the user’s query is related to, since you want to match with concepts even if synonyms or closely related words are used.

WordNet provides this type of functionality [40]. It is basically a dictionary made into a computer-readable format, grouping nouns, verbs, adjectives and adverbs into groups of synonyms, called synsets. These groups are then related to one another through a number of different semantic relations, and have examples and definitions associated with them. Software is distributed along with WordNet that e.g. lets you to find the distance between two synsets.

Though the original WordNet only covers the English language, there have been made versions for many other languages that follow the same format. For example, a Norwegian version called OrdVev exists, available from the national library (Nasjonalbiblioteket)¹¹.

2.2 Existing search function in CKAN

CKAN uses the Apache Solr project¹² for its search functionality, which in turn is based on Lucene. It is essentially a full-text search, which looks for datasets whose metadata include words in your query. CKAN updates Solr so datasets are indexed, and CKAN uses Solr to carry out any searching, while providing the user interface itself. You can see the CKAN search in action in Figure 2.1 on the following page.

Solr includes “Advanced Configurable Text Analysis,” which is designed “to make indexing and querying your content as flexible as possible [4].” Specifically, Solr includes

¹¹<https://www.nb.no/sprakbanken/repositorium#ticketsfrom?lang=en&query=alle&tokens=ordvev&from=1&size=12&collection=sbr>

¹²<https://lucene.apache.org/solr/>

The screenshot shows the Data.gov search interface. At the top, there is a search bar with the text "Search Data.Gov" and a magnifying glass icon. Below the search bar is the Data.gov logo and navigation links for DATA, TOPICS, IMPACT, APPLICATIONS, DEVELOPERS, and CONTACT. A blue header bar contains "DATA CATALOG" and navigation links for "/ Datasets", "Organizations", and a help icon. The search results section shows a search for "rest area" with a magnifying glass icon and a dropdown menu for "Order by" set to "Relevance". Below the search bar, a message states: "You are searching in the list of datasets. Show results in entire Data.gov site." On the left side, there are filter sections: "Filter by location" with an input field "Enter location...", "Topics" with a "Clear All" button and a list of topics including "Local Government (423)", "Ocean (29)", "Climate (21)", "AAPI (4)", and "Disasters (4)"; "Topic Categories" with a "Clear All" button and a list of categories including "Physical and Oceano... (29)", "Water (9)", "Arctic (6)", "Permafrost and Arct... (5)", and "Arctic Ocean, Sea I... (3)"; and "Dataset Type" with a "Clear All" button and a list of types including "geospatial (1770)". The main content area displays "2,011 datasets found for 'rest area'". The first result is "IDOT Rest Areas" with a "State" label, a description, and format options (CSV, RDF, JSON, XML). The second result is "USGS Hydrography (NHD) Overlay Map Service from The National Map - National Geospatial Data Asset (NGDA) National Hydrography Dataset (NHD)" with a "Federal" label, a description, and format options (Esri REST, WMS, HTML, PDF). The third result is "USGS Watershed Boundary Dataset (WBD) Overlay Map Service from The National Map - National Geospatial Data Asset (NGDA) Watershed Boundary Dataset (WBD)" with a "Federal" label, a description, and format options (Esri REST, WMS, HTML, PDF). The fourth result is "USGS Governmental Unit Boundaries Overlay Map Service from The National Map" with a "Federal" label, a description, and format options (Esri REST, WMS, WFS, HTML, PDF). The fifth result is "USGS Geographic Names (GNIS) Overlay Map Service from The National Map - National Geospatial Data Asset (NGDA) Geographic Names Information System (GNIS)" with a "Federal" label, a description, and format options (Esri REST, WMS, WFS, HTML). The sixth result is "USGS NAIP Imagery Overlay Map Service from The National Map".

Figure 2.1: Example of searching using the CKAN interface, here on data.gov.

“[m]any additional text analysis components including word splitting, regex, stemming and more.”

The CKAN project itself writes that “CKAN provides a rich search experience which allows for quick ‘Google-style’ keyword search as well as faceting by tags and browsing between related datasets. [...] all dataset fields are searchable [12].” Furthermore, some fuzzy-matching is available, with an “option to search for closely matching terms instead of exact matches,” likely using the text capabilities of Solr mentioned above.

The faceting feature is an advantage of using Solr. By using it, the user can filter returned datasets by tags, publishers and so on. The user can also see how many datasets there are for the different filtering options. These facets are handy if you know what you are looking for, for example if you know a certain dataset is published by the city of New York. Solr also has the ability to do a geospatial search, using location information to find nearby results. You can see the facets on the left-hand side in Figure 2.1 on the preceding page.

The CKAN search has one weakness, namely the fact that the user’s query must mention words found in the dataset metadata. If the datasets’ metadata were consistently comprehensive and included many synonyms, this would not be a big problem. However, dataset titles and descriptions are often very technical and brief. It can look like they were created by individuals who are very familiar with the subject matter, and who did not have much time for publishing the dataset. This is only natural, considering how the job of publishing datasets is not among the core tasks of a governmental body, but rather just a side-task that has been “forced” upon them to a varying degree.

The end result is that you need to have a similar mindset of the ones publishing the dataset if you are to find them. Especially in domains with a lot of domain-specific terms will there be problems for users who have not acquired the same vocabulary. If you search using a different term than the one used by the dataset publisher, you will likely end up with zero results, even if there are datasets out there that are relevant to your search. The fuzzy matching available does not seem to take into account the *meaning* of words, only how similar they are when written out with letters, so it does not mitigate this problem.

2.3 A previous project on improving the CKAN search

In 2016, as a part of the “Open Transport Data” project and the NTNU subject *TDT4290 Customer Driven Project*, a group of students were tasked with creating:

a user friendly system, based on CKAN technology, that makes it possible to gather the information for all transport related datasets in one location. [...] The system should also provide [a] user-friendly search function based on the semantics of a transport ontology. [22, p. 1]

Due to difficulties working with CKAN, the search solution was implemented as a separate front-end, through which you can make semantic search queries. The backend is implemented as a CKAN extension, which adds a tab to the dataset pages which lets you manually tag them with concepts. Your search query is matched against existing concepts in the ontology, and all datasets associated with either the matched concepts

or their children in the specialization hierarchy are retrieved. If your query is matched with the concept “Vehicle,” all datasets associated with “Vehicle” or any concept beneath Vehicle, like “Car” and “Bus,” are returned.

The project was ambitious, considering CKAN is a quite impenetrable system. It may therefore not come as a surprise that there are some limitations:

- The search result is not ranked.
- No ontology is built into the system, instead you must upload an ontology yourself. Though an ontology developed internally is mentioned, its development is not described and its content is not available in the code repository.
- Though ontologies developed with Protégé could be uploaded, other ontologies in the same file format cause errors when uploaded [22, p. 119].
- The expected format of the ontology is not aligned with best practices for ontologies [2], since concepts are represented as classes without instances and their hierarchy is built using “subClassOf” relationships.

The produced artifacts are superseded by the master thesis written the following year.

2.4 The master thesis which improved CKAN search further

This thesis uses a previous master thesis as a starting point, namely Thomas Hagelien’s thesis on “Ontology Based Semantic Search” [20]. The system he developed was later named *DataOntoSearch*, and is referred to as *DataOntoSearch version 1* in this thesis. An introduction to it is given in this section.

Hagelien’s thesis casts a wide net, attempting to create an ontology for the transport domain, implement a semantic search for open transport data sets and create a system for helping applications interoperate in cases where data fields are given the same semantic meaning in the applications, but the expected data format is different. For the latter part, only a design could be made in time, while for the other two parts, Hagelien created an ontology and a prototype for searching for datasets semantically.

2.4.1 Ontology development

Hagelien did not find the existing Open Transport Data Ontology fit for purpose. Though it may have looked fine to a human, it did not embed much information about how related the different concepts were to one another, due to a very flat hierarchy. He helped develop a new version which is much more useful when used with similarity measures like the Wu-Palmer distance. Additionally, the new ontology has human-readable labels for all concepts in both English and Norwegian, making it possible to use it with dataset metadata and queries of both languages. The process of creating the ontology and an excerpt from it is found in [25].

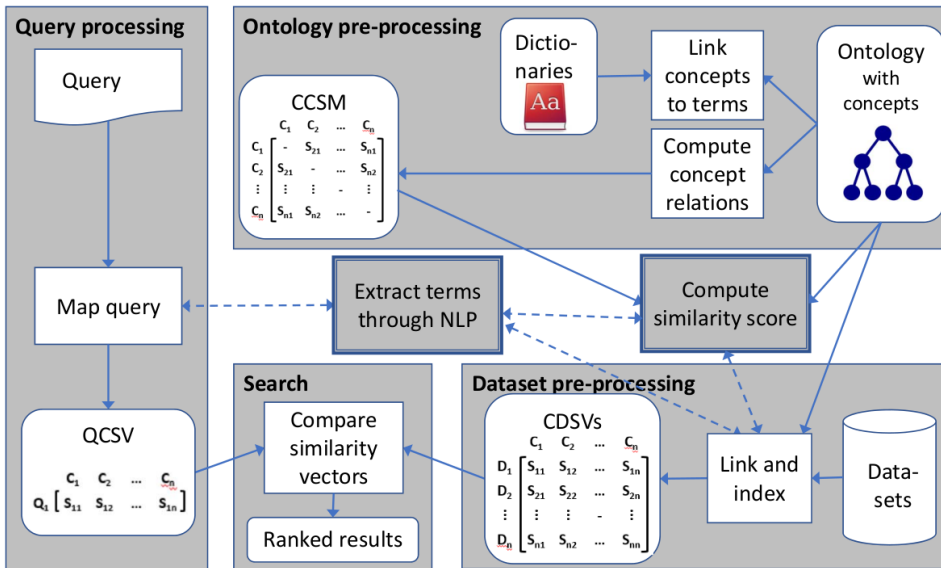


Figure 2.2: Overview over the search process. From [25].

2.4.2 Semantic search

The DataOntoSearch system was developed as a standalone application which is fed datasets and their associations with concepts from other sources. There are two set of associations between datasets and concepts:

1. **Manual tagging:** Manual associations between datasets and concepts, added by domain experts.
2. **Automatic tagging:** Based on the similarity between each dataset's metadata and the concept labels, using NLP methods and WordNet.

They are handled separately, but you can choose which to use when you search. The addition of automatic tagging is one benefit of DataOntoSearch compared to the implementation mentioned in Section 2.3, since many dataset publishers presumably do not have the time or expertise to properly pick concepts. The autotagging procedure is described in detail in [25].

The search procedure of DataOntoSearch version 1 can conceptually be thought of like this:

1. **What concepts are relevant to this query?** NLP and WordNet is used to compare the query's semantic similarity to all concept labels.

The procedure now has a Query-Concept Similarity Vector (QCSV) which contains the similarity between the query and each concept, with each concept corresponding to one column.

2. **What concepts are relevant to each *dataset*?** The manual or the automatic tagging is loaded into the application. This constitutes the Dataset-Concept Similarity Matrix (DCSM), in which the concepts are the columns and each row corresponds to a dataset.
3. **How similar are all the concepts to each other?** For each pair of concepts, their similarity can be found by applying the Wu-Palmer algorithm [50] to the hierarchy of concepts in the ontology. The resulting matrix with concepts as both rows and columns is called the Concept-Concept Similarity Matrix (CCSM).
4. **What *other* concepts are relevant to each dataset?** The method wants to enrich the datasets with concepts that are semantically close to the ones already associated with them, so that you can get a match even if your query did not match the exact same concept. For example, say a dataset *a* is only associated with the concept “Car,” with a similarity score of 1.0. From the CCSM, the procedure can see that the similarity between the two concepts “Car” and “Bicycle” is 0.22. It therefore sets the similarity between *a* and “Bicycle” to $1.0 * 0.22 = 0.22$. The procedure does the same for all the other concepts, updating its Dataset-Concept Similarity Matrix as it goes along.

The procedure now has the Query-Concept Similarity Vector and a Dataset-Concept Similarity Matrix. Both have the same concepts as columns.

5. **How similar is each dataset to the query?** For each row (dataset) in the DCSM, the procedure calculates the row’s similarity to the QCSV using cosine similarity, essentially creating a Query-Dataset Similarity Vector.
6. **What are the most similar datasets?** Using the calculated similarities, the procedure can return the most similar datasets, sorted by decreasing similarity. This is our search result.

Of course, step 2-4 are independent of the query and are in practice done once off-line, before processing queries. The full details of the process and examples of what the matrices look like can be found in [25].

What this process essentially does, is use the ontology’s concepts as a common “language” between the user’s query and the datasets. By “translating” both the query and all datasets to concepts, the method finds what datasets are most similar to the query when expressed as concepts. The use of concepts allows for finding the datasets that are only *indirectly* relevant to the query, hopefully giving DataOntoSearch an edge over the keyword search approach found in CKAN.

2.4.3 Evaluation

Hagelien compared the search process to the original CKAN search using traditional information retrieval evaluation methods. He found that for the seven queries tested, DataOntoSearch version 1 surpassed CKAN for queries using words that were semantically similar to, yet not the exact same as, the words used in the dataset metadata [25]. The evaluation did not consider the result ranking, opting instead to measure the precision and recall after retrieving all matching documents.

Compared to the improvement of CKAN mentioned in Section 2.3, Hagelien's system sports a couple improvements:

- It is independent from CKAN, giving it wider applicability.
- An ontology using SKOS is provided.
- Search results are ranked by relevance.
- The user's query is expanded using WordNet.
- The search engine is able to return datasets related to broader concepts than the ones matched, rather than just the narrower concepts.
- Automatic tagging is implemented.

Related Work

The domain of semantic and ontology based search has existed for some time. Similarly, there is a lot of literature out there regarding how to evaluate such systems. This chapter summarizes some of the work related to semantic and ontology based search and how to evaluate the systems and their search results.

3.1 Semantic based search

This section introduces existing solutions for semantic search.

3.1.1 Google Dataset Search

For a while, Google has used the semantic web to enrich their regular search engine. For example, “[they] provide better discovery and rich content for books, movies, events, recipes, reviews and a number of other content categories [37].” In the beginning of 2017, Google announced through their blogpost [37] that they would like to provide the same capabilities for datasets. The announcement emphasized the importance of data providers to “[publish] structured metadata using schema.org, DCAT, CSVW, and other community standards [37].”

One and a half year later, Google launched their dataset search, available at <http://g.co/datasetsearch> [36]. Although it does not return results from the Norwegian government yet (September 2018), this may be caused by the way the government publish their data:

As more data repositories use the schema.org standard to describe their datasets, the variety and coverage of datasets that users will find in Dataset Search, will continue to grow.

[...]

A search tool like this one is only as good as the metadata that data publishers are willing to provide. We hope to see many of you use the open standards

to describe your data, enabling our users to find the data that they are looking for. [36]

Other actors were quick to adopt Google’s guidelines, with Mark Hahnel, chief executive of a data-sharing company, saying that “By November, all the universities we’re working for had their stuff marked up [10].”

Google’s search solution also finds research articles citing the datasets in question, allowing you to find descriptions and usages of the dataset.

Some drawbacks of this first version of Google Dataset Search solution are:

- No datasets harvested from official data providers in Norway (only versions downloaded and uploaded elsewhere by users).
- No support for Norwegian language.
- Search being one of Google’s biggest assets, the algorithm and procedure is not documented or published anywhere, limiting the knowledge and usage to Google. This is in contrast to open source solutions, which may be used by whoever wants to, and inform the scientific community.
- Formatted metadata from data providers is served directly to users without formatting, leading to some disastrous results in the user interface. An example¹ being a dataset description that includes code examples, which is presented without line breaks on Google and ends up being unreadable.
- No application programming interface provided for automatic use by software.

Many of these problems are intrinsic to any dataset search engine which aims to search among all datasets on the Internet. When limiting the scope to datasets published by one entity, you can customize the user interface and search engine so it fulfils its task better. This does, however, reduce its usefulness as a one stop shop for developers to search for datasets. Other problems are related to Google’s business model, which means they can’t publish the code for everyone to see. Nevertheless, “[e]xperts say that [Google dataset search] fills a gap and could contribute significantly to the success of the open-data movement [10].”

3.1.2 Spatio-Temporal Search

Neumaier and Polleres have created an open source semantic search system for open (government) data, which provides capabilities for finding datasets that pertain to a certain geographical area and period of time [32]. Utilizing the breadth of information available on the Semantic Web, their system lets the user filter based on actual geographical names and named periods of time, like “Oslo” and “the cold war,” instead of simply using coordinates and start and end dates.

Their system has to do several tasks to accomplish its overarching goals:

¹<http://web.archive.org/web/20180920132203/https://toolbox.google.com/datasetsearch/search?query=bus%20departure&docid=H84sSCiT5CHKvdKOAAAAA%3D%3D>

1. **Consolidate data from the Semantic Web:** Several sources are used to create a knowledge graph which can be used to map queries and datasets to spatial and temporal entities. For instance, something as “simple” as national postal codes will require information from many different sources, since no single comprehensive data source include them for all countries. Different sources may also provide different details about the same entities.
2. **Label datasets with spatio-temporal information:** In much the same way the autotagging of DataOntoSearch links datasets to concepts, this system needs to enrich the datasets with links to the spatial and temporal entities extracted in the previous step. This is done by looking at the dataset metadata, as well as the dataset contents.

A number of different heuristics are used to check whether a column contains geographical or temporal information, and when such a column is detected, its contents can be used to determine what spatial and temporal entities the dataset should be linked to. This can be challenging due to ambiguities, like four-digit postal codes potentially being misinterpreted as years, or the text “Norwegen” in a German dataset (“Norway” in English) being taken to mean the small region in Germany, not the country. Nevertheless, using the dataset contents is a promising direction that is especially useful when the dataset metadata themselves are lacking.
3. **Export RDF:** All the information gathered in the preceding two steps is made available as RDF through a SPARQL endpoint, including taggings of datasets and their individual table columns and cells.
4. **Provide search:** Both a graphical user interface and an API is available for use. There are separate fields for entering a geographical location and keywords and selecting a time period, all of which are used to filter the datasets. Autocomplete is provided, so the user selects an entity the system knows. When you query using the graphical user interface, you also get to see an excerpt from the dataset contents, where the geographical location you chose is highlighted.

Neumaier and Polleres evaluated the dataset labelling system to check its correctness. They found that for a random sample of datasets, the labeller’s average precision was 86% and its recall was 73%, leaving some room for improvement. The results depended on the dataset’s country of origin, however, with some countries having near perfect scores and some others not scoring so well.

The search itself and its graphical user interface was not evaluated by Neumaier and Polleres, though it is available online² and seems to be a prototype.

3.1.3 OntRank

Xiaolong Tang et al. experimented in 2013 with a way of adapting generic RDF data to fit with classic information retrieval methods, while adding an ontology factor to the ranking algorithm [45]. They found that existing approaches either were unsuitable for

²<https://data.wu.ac.at/odgraphsearch/>

novice users since they required knowledge of the underlying semantic data and a query language, or had poor search results.

Since traditional information retrieval operates on documents, the RDF data must be transformed to fit this structure of documents. Xiaolong Tang et al. do this by looking at all the entities that appear as subjects in the RDF graph. For each such entity, its corresponding document is created by including all triples that have that entity as the subject. This is similar in nature to documents which have different properties attached to them, so the BM25F algorithm is chosen.

Two indices are created, one for the documents themselves and one for the ontology, which is based on information from DBpedia. Due to the sheer size of graphs obtained from linked data, Xialong Tang et al. focus on efficiency and scalability in the indexing procedures, adopting the Map-Reduce pattern to effectively utilize concurrency. They use ORDPATH as the hierarchical coding scheme, which handles updates well and gives an easy way of calculating the path distance between two items in the hierarchy.

Xialong Tang et al. introduce a ranking algorithm called OntRank, which combines the BM25F algorithm with a new factor called RO. It is based on the semantic similarity between the query and the document, as indicated by the distance of the path between them in the ontology.

When evaluating OntRank against just BM25F alone, they found that the first 15 documents retrieved were relevant to a greater degree with OntRank. Other measures showed only slight improvements, likely because there are some limitations in the ontology and its classification. Theories and algorithms are for instance just being classified as owl:Thing since no specific classification exists, and some entities are misclassified. Xialong Tang et al. believe that as the quality of the underlying data improve, the OntRank algorithm will show greater improvements.

3.1.4 Dataset search using semi-structured query patterns

Buranarach et al. explored an approach that lies between the full-text search of CKAN and a structured search like SQL or SparQL, using three different semi-structured query patterns [8]. Furthermore, it looks into the datasets' content when searching, helping users find the data they need instead of having to go through dataset descriptions.

As an example, you have the pattern “⟨class⟩ ⟨property⟩ ⟨value⟩.” By searching for “income province bangkok,” you will get dataset rows about *income*, for which *province* is *Bangkok*. The other types of patterns are “⟨property⟩ ⟨subject⟩” and the same, just with the opposite order. Here, searching for “telephoneNo Rajini School” would search for datasets with the *telephoneNo* column, and specifically find the row for *Rajini School*. As you can see, the system presents not only datasets as documents, but presents their contents in a table format. If the user's query doesn't match any of the patterns, the search falls back to a regular keyword search.

It could potentially be very awkward for the user to write queries in these formats when not knowing what to search for. Therefore, search suggestions are implemented, giving the user options for potential classes, properties, values and subjects, based on what pattern is found to be used. By following the autocomplete suggestions, users are guaranteed a query which fits with one of the query patterns.

Buranarach et al. developed a prototype system which used ten datasets from the Thai government's open data portal, Data.go.th. The classes, properties, values and subjects had to be extracted from the dataset contents in question in order to build an index, so that the search suggestions could be given on the fly. RDF is used internally to represent the metadata, and the user's query is parsed and made into a SparQL query as a part of the search procedure.

Even with the limited number of datasets used in the prototype, Buranarach et al. encountered scalability problems. The index grows quite large, hosting over 160 properties and 25 000 term relations. They found that “[t]his can greatly reduce the performance of the system in making query suggestion[s] [8].” Furthermore, the format of the datasets themselves can give some confusion, e.g. with column headers like “TelNo” that are not natural for the user, and it is desirable to support queries with more than just one property-value pair.

Buranarach et al. do not present any evaluations of the system. It could have been interesting to see if and how end users would adapt to the semi-structured query patterns in a usability test, for instance.

3.1.5 Ontology-based query improvement

Xu and Li created a search engine which uses an ontology to help the user improve their queries before giving it to a keyword-based search engine [51]. Their ontologies are similar in nature to that of DataOntoSearch version 1, in that it is a hierarchy of concepts organized with “broader” and “narrower” relationships, though they also use “equivalent” relationships and define properties of some concepts.

When the user attempts to search, their query is matched with concepts. Based on the concepts' relationships in the ontology, related concepts are found and shown to the user, which can either click another concept to browse *its* related concepts, or perform a regular keyword-based search using the concept. Though the system uses ontologies and semantic technologies to help the user find the right query to make, the actual searching is left to an off-the-shelf system and is done without involvement of the ontology.

A case study was done by developing an ontology for the computer science domain, based on two university courses taught by one of the authors. Xu and Li argue that the system benefits users without intimate domain knowledge, since they can arrive at the concept they wanted to search for by searching for the related concepts they actually remember. The system's capabilities for finding related concepts is also quite advanced, though it does not seem to include any ranking functionality. That said, they do not present any evaluation of the system. It might have been interesting to know what students attending the courses thought about the system's usefulness.

3.1.6 Other semantic search engines

There have been a lot of attempts at creating a search engine for searching the linked data on the semantic web. Aidan Hogan et al. [24] mention a couple examples, like the early attempts Ontobroker [14] and SHOE [23], document-centric approaches like Swoogle [15] and Sindice [39], and entity-centric approaches like Sig.Ma [46], Watson [13] and the Falcons Search engine [11].

A different search engine for searching among RDF entities in the Semantic Web is the Semantic Web Search Engine, abbreviated SWSE [24]. It adopts a holistic approach which aims at giving the average user a way to access linked data, which brings its own challenges due to the scale of the semantic web, mistakes and noise in the data and needing to make this usable for people without knowledge of semantic technologies. The idea is to collect information about an entity from different sources and present it unified, instead of simply presenting each individual source of information the way document-centric systems such as Google does. Interestingly, they echo the sentiment that “the quality of data, that a system such as SWSE operates over, is perhaps as much of a factor in the system’s utility as the design of the system itself [24].” Though their system proved able to handle the scale of the semantic web, they leave regular information retrieval evaluation methods like precision and recall and usability testing as possible future work.

Morales and Melgar present a systematic literature review of architectures of semantic search engines in [31]. The use of ontologies is one of the topics they investigate. They found that “[d]omain ontologies are mostly used, which seems to be a pattern across architectures.” Though some approaches used general purpose ontologies like WordNet as well, they ended up using domain specific ontologies to better represent user concepts. The roles of ontologies, Morales and Melgar found, are diverse, “but most of the selected architecture[s] use them as a way to classify and express relationships among key concepts [31].”

When DataOntoSearch version 1 was presented, Shanshan Jiang et al. noted that “[t]o the best of our knowledge, there are no reports on applying semantic search on open data, in particular, for the widely used open data platform or portals [25].” As far as I can tell, no new reports on the topic have been published since then, apart from the spatio-temporal search described in Section 3.1.2.

3.2 Evaluation approaches of semantic search

Ali and Beg reviewed approaches to evaluating non-semantic web search, categorizing them into eight different categories [1]. Among them are relevance based methods, representing information retrieval methods like recall and precision; ranking based methods, putting focus on the ranking of the top 20 items; user satisfaction methods, measuring users’ satisfaction with the system; and automatic evaluation approaches, which attempt to eliminate the need for human evaluators.

As summarized by Elbedweihy, there are two categories of evaluations for information retrieval systems [16]:

- **System-oriented evaluation:** is concerned about the search results to certain queries, evaluating the system’s ability to fetch relevant documents and rank them highly. This type of evaluation “simulates” the user, and its validity is therefore dependent on how well the “simulation” matches up with real users’ behaviour. It is popular since it can be automated to a high degree once relevant documents have been decided, lending itself well to repeated evaluations.
- **User-oriented evaluation:** is more concerned about the entire user experience. This is achieved by testing the system in a more realistic setting, and also taking into

consideration the experience of formulating a search query and browsing the results. Though the results may be more valid, this kind of evaluation requires a lot of effort.

These two approaches are described in more detail in the next sections.

3.3 System-oriented Evaluation

System-oriented evaluation encompasses methods usually associated with information retrieval. Instead of having real users spend time with the system, their queries are “simulated” and ran systematically, and the users’ satisfaction with the results are “simulated” by having judges decide which results were relevant. System-oriented evaluations are generally set up this way:

1. Decide the dataset to use for search.
2. Formulate queries to be performed on the system under test.
3. Run the queries with each system, recording the result.
4. Depending on the evaluation method, decide what documents or entities are relevant to each query, either for returned documents or the entire document collection.
5. Calculate scores.

A common thread through this process is the wish to emulate the real world as closely as possible. For instance, the chosen dataset should, according to Elbedweihy, include at least 100 million triples in order to be acceptable, have triples of varying origin and quality, and be up-to-date with current standards [16]. Similarly, queries should be representative of queries users would make, and the relevance assessment should be guided enough for the assessments to reflect what a user might think.

Even if you make sure to be as close to the real world as possible, system-oriented evaluation methods will not align perfectly with how users would use a system. The presentation of the system, instruction for search and guidance given while writing the search query all affect the user experience, so system-oriented evaluation is no substitute for user-oriented evaluation. What it provides, is a way of testing often or testing many systems. For example, it can help you test a system during development, allowing you to see how much a new change improved the search engine. Of course, this requires that the dataset, queries and relevance assessment has been done, all of which are manual processes, and to a varying degree labour intensive.

Some of the measures available are described below. They all vary in what parts of the search results they use, what kind of relevance assessment they require, and what their score can tell us about the system being tested.

3.3.1 Precision and recall

Precision is a measure of how many of the retrieved documents are relevant [29]. For example, if a system retrieved 50 documents, out of which 20 were relevant, its precision

would be $\frac{20}{50} = \frac{2}{5}$. A high precision score indicates a system which avoids retrieving irrelevant documents, while a low score is indicative of a system that retrieves a lot of noise.

Recall is a measure of how many of the relevant documents were retrieved. For example, if there are 60 relevant documents in the collection, out of which only 20 were retrieved, the recall would be $\frac{20}{60} = \frac{1}{3}$. A high recall score indicates that the system retrieves a complete result, so you can trust that you are not missing out of any relevant documents. A low recall score means that many relevant documents remain inaccessible to the user.

Precision and recall can often be thought of as opposite ends of a spectrum. A system with a high precision score might be leaving out a lot of documents, including relevant documents, resulting in a low recall score. Similarly, a system with a high recall score might be retrieving a lot of documents, including irrelevant documents, affecting precision negatively. Of course, a perfect system would retrieve all relevant documents and nothing else, resulting in perfect precision and recall scores. It is possible to combine precision and recall into one measure called the **F1 score**, by taking the harmonic mean of precision and recall.

Precision only requires a **relevance assessment of the documents retrieved** by the system. On the other hand, recall requires that **all documents in the entire document collection are assessed**, so you can know how many relevant documents there are. Both for this reason and the fact that most users only care about getting their question answered, not seeing all possible documents answering their question, precision is often used and recall is dropped when using large document collections, as is the case for web search and semantic search. Alternatively, a less resource intensive way of deciding relevant documents can be used, like using the documents retrieved by multiple systems. This could possibly decrease the measurement's validity, however, by not "simulating" the user well.

Precision and recall, in their unaltered form, do **not** take into consideration the **ranking of the results**. A system that ranks the relevant documents at the bottom will receive the same score as one that ranks them at the top.

3.3.2 Precision at k

While the precision measure looks at the entire collection of retrieved documents, users typically do not care about what is retrieved beyond the first page of search results. To better simulate this behaviour, the **precision at k** measure can be used to calculate the precision when the top k documents have been retrieved.

Unfortunately, this is not a stable measure, since it is highly dependent on the cut-off value k relative to the total number of relevant documents. If one query has a hundred more relevant documents than another query, and the cut-off value is fixed for all queries (as is typically the case), then we can expect the score to be much better for the first query compared to the second, even if random documents are retrieved.

There is also the unfortunate fact that if the number of relevant documents is lower than the cut-off value, a perfect score is not possible.

3.3.3 R-Precision

R-Precision can be thought of as *precision at k*, except k is set dynamically to the number of relevant documents for each query. Thus, if there are 60 relevant documents to a query, then R-Precision is the precision score when examining the top 60 retrieved documents.

This measure avoids the problems of *precision at k*, since it scales with the number of relevant documents, and a perfect score can be achieved for any query when the system retrieves only relevant documents.

3.3.4 Mean Average Precision

All measures described until now only calculate the precision and/or recall at a certain point, whether it be when all documents are retrieved or only a set number. However, it can be beneficial to measure precision at multiple points, to better evaluate a system's capability to rank relevant documents highly.

Consider a single query. The **Average Precision** is the average of multiple precision scores. They can be measured either at different levels of recall (like 0.0, 0.1, . . . , 0.9, 1.0) [16] or once every time a relevant document is retrieved [29]. The precision for relevant documents that are never retrieved is set to 0.

The average precision can itself be averaged over multiple queries, in which case it is called **Mean Average Precision**, abbreviated MAP.

MAP is a common measure, which gives a good idea of how well a system ranks its results. It is also a stable measure, and is well equipped to separate the best systems from the rest.

3.3.5 Receiver Operating Characteristics (ROC)

A ROC curve is an alternative to MAP. It shows how the recall score, mapped to the Y axis, increases as more and more irrelevant documents are retrieved, mapped to the X axis. If the curve starts with a steep climb on the left side, then many relevant documents are retrieved before irrelevant ones. On the other hand, if the graph raises slowly then many irrelevant documents are spread among the relevant documents retrieved.

As a system gets better and better, the ROC curve starts with a climb that gets steeper, earlier. It therefore stands to reason that the area under the ROC curve can be used as a good measure. In practice, the area under the ROC curve is often used in fields other than Information Retrieval (IR), while MAP is more common in the IR field.

3.3.6 Normalized Discounted Cumulative Gain

This is a measure using *graded* relevance assessments, granting greater granularity than the simple relevant/irrelevant scheme used by the measures described up until now. Such graded relevance assessments let you decide that one document is a little bit relevant, while another one is very relevant to the query.

Cumulative Gain (CG) is the simplest way of measuring using graded relevance scores. The cumulative gain at a level i is simply the sum of the relevance score of each document that has been retrieved when i documents are retrieved. When you retrieve one

more document, its relevance score is added to the cumulative gain at the level above. As opposed to precision and recall, the cumulative gain is not normalized by the number of retrieved or relevant documents. However, just like precision and recall, it does not care where in the ranked list relevant documents appear.

Discounted Cumulative Gain (DCG) adds a discount function to Cumulative Gain, so that the relevance scores from low-ranked relevant documents contribute less to the total than the relevance scores of top-ranked documents. Thus, more emphasis is placed on the documents ranked higher. The discount function used is defined as $\frac{1}{\log_b i}$, where b is a variable you can use to adjust to what degree lower-ranked documents are discounted.

Normalized Discounted Cumulative Gain (NDCG) is to DCG what R-Precision is to Precision at k , though the method used is not the same. An ideal set of DCG values is created, representing what the result of a perfect information retrieval system would be. The actual DCG for the system being tested is then divided by the ideal DCG values, so that a perfect system achieves a score of 1.

3.3.7 Best Practice

Elbedweihy et. al. describe some best practices for evaluating semantic search systems [16]. They stress the importance of user-oriented evaluation, especially for semantic search applications, and recommend an approach using both types of evaluations. The system-oriented evaluation recommendations are described here.

Choice of dataset

The aim is to have a realistic dataset, which can function as a stand-in for the set of data that would be used in a real-world situation. In the case of a generic system used for a single domain, its dataset should contain over 100 million triples, while open-domain systems be at least 10^9 triples large. They should ideally vary in source, origin and quality, to better emulate the messy nature of real-world datasets. They should be up-to-date with current standards and technologies.

Choice of queries

For the choice of number of queries, “between 50 and 100 queries would be acceptable.” They should represent genuine information needs from users, being based on interviews with users or search logs. However, using search logs is very difficult due to semantic search not being widely adopted by regular users. Instead of simply writing down the queries, it is best to record some more information along with the query, like the information need that led to the query, so that it is easier to determine what is relevant afterwards.

Relevance assessment

Instead of assessing all documents, the results from different systems should be put together to form a pool of the top K results. This pool is then evaluated by human judges to determine what is relevant and not. An effort must be made to avoid any biases that may occur from aspects like the choice of judges and the order documents are presented in.

Evaluation criteria

Measures usable for ranked results and graded relevance assessment should ideally be used. NDCG is, in that case, a good candidate. However, the number of queries and results assessed per query must be taken into account when deciding what measures to use, since the stability of measures depend on those factors.

3.3.8 Crowdsourcing relevance assessment

While expert judges have typically been used for determining the relevance of retrieved results for different queries, a new approach was examined by Blanco et. al. in which the task is split up and sent to workers participating in the Amazon Mechanical Turk [6]. The lack of expertise is made up for by drastically increasing the number of judges, with overlapping relevance assessments, and by asking for a relevance assessment of items already assessed by experts, allowing the system to filter out judges not fit for the task. This approach makes for more reliable and repeatable results in a scalable manner, avoiding the pitfalls of using few experts whose individual opinions are given a high weight.

Before using a solution like this, researchers much give proper thought to the problems inherit in using a system like the Amazon Mechanical Turk, in which workers are often paid less than the minimal wage in the US [21]. Workers are not compensated for the down time in-between assignments or their use of a computer and internet connection. Since they are essentially their own business owners taking on work from the Mechanical Turk, they are not entitled to any health care either. Researchers must think twice before engaging in a system which abuses workers, especially considering the increasing public awareness of this problem, as demonstrated by the recent Uber and Lyft strike [38] and critical press articles [42, 43].

3.4 User-oriented Evaluation

As previously mentioned, user-oriented evaluation, also referred to as usability testing, considers the entirety of the user's experience with the system under test. Before we dive into the methodology used, we will take a look at what usability actually is. At the end, we go through a related study which uses some of the discussed methods.

3.4.1 Defining Usability

There are multiple ways of defining usability. According to Jakob Nielsen, "Usability is a quality attribute that assesses how easy user interfaces are to use [33]." He lists five quality components for usability [33]:

- **Learnability:** How easy it is for users to learn and use the system on their first encounter.
- **Efficiency:** How efficient the users are once they have learnt the system.
- **Memorability:** How easy it is for users to re-learn the system after a period of not using it.

- **Errors:** The rate of errors users make, the severity of their consequences and how easy the users recover from them.
- **Satisfaction:** How pleasant the system is to use.

Nielsen argues on page 26 of *Usability Engineering* [34] that “[o]nly by defining the abstract concept of ‘usability’ in terms of these more precise and measurable components can we [as a discipline systematically approach, improve and evaluate usability].”

To further define the word “usability,” he relates it to the concepts of “utility” and “usefulness” in [33]:

- Definition of **Utility** = whether it provides the **features you need**.
- Definition of **Usability** = how **easy & pleasant** these features are to use.
- Definition of **Useful** = **usability + utility**.

It follows that systems without the needed features are not useful, no matter how easy and pleasant their design is. Similarly, a feature might as well not exist if it is never discovered by users, or users are unable to use it well due to problems with the design.

When you try to define a concept, it may be wise to check in with the established international standards. The International Organization for Standardization (ISO) has defined two standards related to usability, namely ISO 9241-11 and ISO/IEC 25010. They describe respectively the concept of usability, and software quality models for quality evaluation, in which “‘usability’ is a subset of *quality in use*,” according to [44] who cites ISO/IEC 25010. ISO 9241-11 defines usability as follows, according to another citation in [44]:

The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use

An important take-away from this definition is how a product’s usability is measured in relation to a certain set of users trying to achieve a certain set of goals, all in a certain context of use. Consider for example a mobile application showing a map. This application might have a very different usability for young adults using it to find a nearby café while at home, compared to seniors trying to find a suitable trail towards a mountain top while squinting at the mobile screen in the sunlight. If the company is trying to improve the application’s usability for the second scenario, they would be ill-advised to run a usability study using young participants in the meeting room of an office (though they might discover some relevant problems in the design). The measurements from such a study could not be used to evaluate the usability for the senior scenario.

3.4.2 Evaluating usability

The topic of evaluating usability is a large one, which can only be briefly summarized here. Specifically, topics like the difference between formative and summative evaluations, methodological pitfalls, methods used and how to run a comparative usability test are briefly covered.

Two types of evaluations

You can separate usability tests into two categories, depending on what type of evaluation they aim to perform [34]:

- **Formative evaluation** helps find out what works and what does not, providing valuable input to future design iterations. This is often called qualitative evaluation.
- **Summative evaluation** measures the usability of the system (or systems) tested, as a part of assessing the system's quality. This is often called quantitative evaluation.

Formative evaluations are useful to do before you start to design or while you are designing a system [34]. They give you an overview of problem areas as well as specific problems you need to solve. On the other hand, summative evaluations give you a way of comparing several systems, and let you validate whether or not a certain system meets its usability requirements. They are therefore typically performed when you have a system to test, such as when deciding what off-the-shelf system you want to use, or when validating a finished system.

Ensuring trustworthy results

When you perform a usability test, you must be careful about the test's **reliability** and validity. "Reliability is the question of whether one would get the same result if the test were to be repeated [34]," and is problematic for usability tests because the raw results depend a lot on the individual test participants. Users have a varying amount of experience with computers, to name just one factor. To combat this problem, you need to test with more users and use statistical methods to discover the extent to which you can draw conclusions based on the collected data.

The test's **validity** is closely related to how the measured usability is dependent on the users, their goals and their context of use. When assessing a test's validity, you are finding out to what extent the test results reflect the actual usability of the product in use. Or more precisely, whether the test tests what you wanted to test. If you intend to test the usability of a product for office workers, but only test using students, the expected validity would be poor. Similarly, if you intend to test the usability difference between the Iphone and Android, but test using a small mobile phone for Android and a big mobile phone for Iphone, your test results might tell you more about the impact of the phone's size than the impact of its operating system on usability.

Elbedweihy et. al. [16] also address the issue of reliability and validity when considering summative evaluations of Semantic Web systems. They found that "inconsistencies in the dataset, as well as naming techniques used within the Semantic Web community, could affect the user's experience and their ability to perform the search task [16]." This could vary between users and datasets, thus affecting the test's reliability. To combat this, they argue that a balance must be struck between choosing a dataset that properly represents a realistic situation, and choosing a dataset which users understand. Furthermore, when considering the number of test users to recruit, they argue that "a number ranging between 8 and 12 subjects would be acceptable."

As for validity, Elbedweihy et. al. note that “[m]any systems developed within the Semantic Web community have been evaluated by experts [16].” This affects validity negatively, since many of these systems wish to become mainstream and find an audience of causal users. However, using both experts and non-experts can give some interesting findings on how the system accommodates the two user groups and how their requirements differ.

Planning the test

Before you start recruiting users for the test, you must create a test plan. Putting it together involves making some important decisions on the intentions and contents of the test, such as:

- What do you want to get out of this test?
- What users will you test with?
- What tasks will they perform?
- When and where will the tests take place?
- Who will run the tests?
- What is the budget for the tests?

See *Usability Engineering* [34, p. 170] for a complete list of questions.

Usability testing methods

As previously mentioned, there are two ways of evaluating usability: formative evaluation and summative evaluation. The two types of evaluations favor different kinds of methods, with formative evaluation favoring methods like thinking aloud, and summative evaluation favoring methods like performance measurement.

The **thinking aloud** method involves asking the user to “think aloud” while performing the task. This is valuable because you can see where the user’s mental model is different from what was intended from the system. You also learn how the user interpret the different parts of the interface. Of course, observing the user is just as import as listening to the user, since users generally don’t know why they did what they did, and may make incorrect rationalizations after-the-fact. Some drawbacks of this method include how test participants find the thinking aloud method to be unnatural, and the fact that they might realize shortcomings in their reasoning when putting it into words, which they otherwise would not have realized, had they sit quiet in their office. Thus, you cannot trust that their behaviour properly represents how a user would interact with the system in the wild.

The **performance measurement** method measures how fast and with how many errors users complete tasks. You can for instance use a stopwatch which you start at a clearly defined time when the task begins, and which you stop when the user has reached their goal, again something which should be clearly defined in the test task description. The definition of what counts as an “error” should also be agreed upon. Pilot tests should be used to reveal inconsistencies and ambiguities in these definitions.

Yet another tool, the **System Usability Scale (SUS)**, is useful for measuring the subjective usability of a system in a “quick and dirty” way [7]. It is a questionnaire consisting of 10 questions, which the test participant answers using the *Likert* scale, a scale between 1 and 5 where 1 is “Strongly disagree” and 5 is “Strongly agree.” They should answer it right after having used the system under test, but before any debriefing. It can be used in conjunction with both the thinking aloud and performance measurement methods, since it measures the subjective aspect of usability. This type of survey is typically how you learn how pleasant the user found the system. Other scales can be made, to capture the information you are interested in.

Comparing different systems

While usability testing can be done with just one system in isolation to elicit requirements for new versions or evaluate its fitness for purpose, you can often learn more by testing with multiple systems. Doing so also allows you to compare them with each other in a scientifically reliable way, avoiding differences caused by different users, different test setups and other subtle differences. Since usability is not an absolute metric, you cannot easily compare across studies.

When testing multiple systems, you must choose between “between-subjects” and “within-subjects” testing [34, p. 178].

- **Between-subjects testing** exposes each test user to a single system only. This means that individual variations in users may impact how the systems compare to one another, for example if one system is tested by users with more computer experience. It is necessary to test with more users and assign these users to systems in a way that avoids bias.
- **Within-subjects testing** puts each test user through the same systems. This means that they will be more familiar with the task and its domain by the time they test the second (and third) system. It is therefore necessary to divide users into different groups, where each group tests the systems in a different order. That way, no single system is placed in a “favorable” position more than the others.

According to an article by the Nielsen Norman Group, “Competitive usability evaluations are a method to determine **how your site performs in relation to your competitors’ sites** [41].” They advocate using within-subjects testing with the thinking aloud method, using 2 or 3 different sites per test user [28]. At the end, the test user should be asked to compare the sites. They also argue that knowing what system has the best usability is not as useful as learning “what worked and what didn’t across designs [41].” Knowing that your design is 30% worse than your competition doesn’t help you as much as knowing that the competitor’s sign up form was much less painful than yours because of e.g. timely feedback.

Similarly, Elbedweihy et. al. recommend asking the user to complete a comparative questionnaire after evaluating all systems rather than relying only on individual questionnaires answered after each system [16]. The reason is that the latter cannot be trusted as much when comparing across systems, due to the time gap between the questionnaires and the lack of any frame of reference.

3.4.3 Evaluating Usability of Semantic Search Interfaces: An Example Study

Kaufmann and Bernstein [26] looked at so-called “Natural Language Interfaces” (NLI) enabling users to query the semantic web using natural language rather than a formal language like SPARQL. More specifically, they created four systems specifically for the evaluation, and evaluated their user experience by running a usability study, aiming to validate their hypothesis that a guided natural language query language provides the best user experience (compared to a free-form natural language query language and formal query language).

This is a competitive usability study, using within-subjects testing with the performance measure method. Kaufmann and Bernstein reference Jacob Nielsen’s *Usability Engineering* [34], along with a German article about user-focused application benchmarking. Their approach to the experiment was to have each test user:

1. Read general instructions about the experiment, printed on a paper to ensure there were no differences in the presentation given by experiment runners.
2. Read instructions about the first query language.
3. Perform the tasks with that NLI.
4. Fill a SUS questionnaire.
5. Read the instructions for the next query language, and so on.
6. Answer a comparative questionnaire after having used all systems, asking what system they liked the best and least, and why.

They ran usability tests with 48 participants, which were given monetary compensation for their participation. The participants were recruited from the general population, not just students or computer engineers. The number of participants allowed them to test each order of NLIs twice, so they could account for bias that might occur from the test order. A number of statistical measurements were used to answer the research question, comparing the SUS scores between the NLIs as well as the test users’ performance. They also looked at free-text comments about the NLIs, gathered from the comparative questionnaire given to users after they had tried out all interfaces.

The four NLIs created and tested are shown in Figure 3.1, and were:

- a free-form NLI
- a somewhat guided natural language question/answer interface
- a very guided NLI
- a graphical SPARQL query editor

The somewhat guided NLI was liked the best by users, while the SPARQL query editor was liked the least. The absolute free-form NLI and the very guided NLI were rated about the same, midways between the two others. Interestingly enough, users thought they had

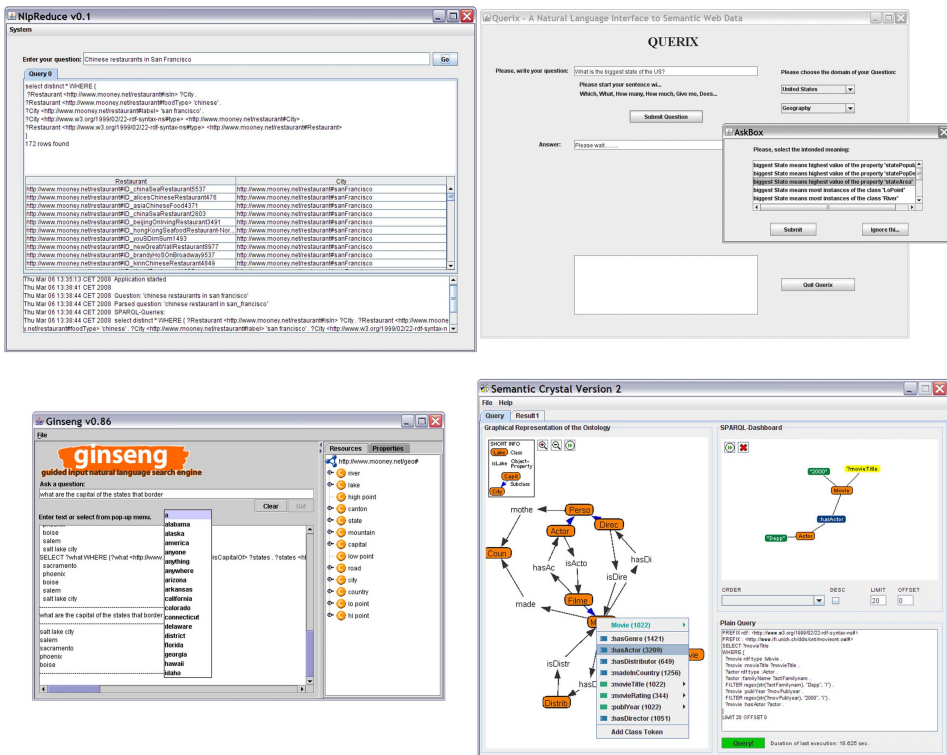


Figure 3.1: The four interfaces created in [26]. Upper left is the free-form NLI, upper right the somewhat guided NLI, lower left the very guided NLI and lower right the graphical SPARQL editor. The original screenshots are property of the paper’s authors.

arrived at the correct answer more often than they actually did with the somewhat guided NLI, probably because the interface gives off an impression of being an oracle, giving the user no opportunity to cross-check its conclusion, while the other interfaces are closer to traditional search engines.

One of the more applicable wisdoms from Kaufmann and Bernstein comes from the contradictory user comments. Some praised the free-form NLI for being similar to common search engines, while others criticized it for being “too relaxed.” The other NLIs received similarly contradictory comments, suggesting that different users preferred different kinds of interfaces. Kaufmann and Bernstein therefore suggest that “we should consider interfaces to Semantic Web data that offer a combination of graphically displayed as well as keyword-based and full-sentence query languages [26].”

As for their hypothesis, it seemed to be confirmed, though users had a greater (actual) success rate with the free-form NLI than the somewhat guided NLI.

As with any study, there are some problematic aspects here. Specifically:

- **More than one variable:** Not only did the query language differ between the different NLI. The interface for writing the query and the result display were significantly different, with the free-form NLI displaying results using their unreadable RDF URI,

while the favourably rated NLI gave back the answer in plain language. This being an experiment, it is not possible to know whether it was the different user interface or the different query language that made users prefer the somewhat guided NLI. The authors argue that “Given the qualitative statements we doubt that the impact is so profound that it solely explains [the somewhat guided NLI’s] lead [26],” though it is difficult to see how the users’ comments support this conclusion.

- **Limited types of questions:** Users were tasked with answering purely factual questions, like “number of lakes in Florida? [26].” Other types of questions, like “how to fix permission denied when mounting a cifs” might have resulted in a different NLI being preferred by users.

Chapter 4

Overall design

Now that this project’s background and the related work done in the field of semantic search and open data have been introduced, it is time to look at the work specific to this thesis. In this chapter, you will learn about the goals of this project, along with their motivation and the overall approach.

4.1 Motivation

Discoverability is a prerequisite for achieving the benefits touted by open data advocates, like improving accountability and generating value for businesses and potential start-ups. If the general public are not made aware of the data’s existence, or they are unable to find the data, it does not really make any difference whether it exists or not. To put it another way, the data can hardly be used for any good, if it is made accessible “in the bottom of a locked filing cabinet stuck in a disused lavatory with a sign on the door saying *Beware of the Leopard*¹.”

Open data has been made public by governments and different municipalities through the use of dataset archival solutions like CKAN. As explained in Section 2.2, though its search is powerful, it is only as good as the metadata that publishers have added to their datasets. Given how the dataset publishers do not experience the downsides of poor metadata, since they only publish datasets and rarely will need to consume them through the publishing solutions, it may not be surprising that the metadata is not as comprehensive as we desire. If we wish to make datasets discoverable, we must find ways of letting users explore and find datasets, even when they don’t know the exact words used by the publisher.

As explained in Chapter 2, some effort has already been put into solving this problem. The customer-driven project in Section 2.3 showed how using an ontology can help finding datasets tangentially related to the user’s query, even if it was not labelled with the exact same concept the user wrote. Thomas Hagelien’s DataOntoSearch version 1 in Section 2.4

¹Douglas Adams, *Hitchhiker’s Guide to the Galaxy*, 1979

provides the foundational basis for this thesis, and demonstrated how the approach can be expanded so that the ontology embeds more knowledge, have the search be independent of CKAN, alleviate the need for manually tagging datasets with concepts and improve how related concepts and datasets are retrieved and ranked.

DataOntoSearch version 1 is still far from perfect. Some of the weaknesses of the process and the product are:

- W1. **Evaluation needs improvement:** At the end of Hagelien’s thesis, we know how the system performed with a few queries, but we do not know what actual users would think of the system. The queries used may not represent how users would approach a system like this, so their validity may not be so good. In addition, there is no testing of DataOntoSearch’s ability to rank the retrieved datasets, resulting in a focus on the cut-off point for retrieving datasets which is not so interesting in an online search context. There are also more systems available to test now, with the introduction of Google Dataset Search.
- W2. **Irreproducible system setup:** Though the code for the system was made available on GitHub, important parts were missing. Some of those parts are available in the thesis PDF, though they extend beyond the margins and are thus inaccessible. There are also no instructions on how to get the system to run. Ironically, the system meant to make published datasets more accessible to developers, is itself *technically* published, yet completely impenetrable.
- W3. **Separation from CKAN:** The system is separate from CKAN, making it more difficult to apply to the CKAN project. On the other hand, this makes it possible to use the system with more than one CKAN instance, and it may also interoperate with any system or dataset catalog that uses DCAT. It is still desirable to integrate it in some way, since users cannot be expected to use an external system for tagging or searching.
- W4. **Some concepts cannot be matched with:** Some experiments show that in version 1, there actually exist concepts that cannot be matched with a user’s query. Some of this may be due to the limits of WordNet, but the process of converting text to WordNet synsets is different for concept labels and the user’s query and autotagging. Thus, even if you write the concept label verbatim, the system will not match you with that concept, particularly for concepts consisting of multiple words. This may also be caused by labels not having corresponding entries in the WordNet word catalog.
- W5. **Only ontology-based semantic search is used:** The system is only able to search through concepts. Aspects that are not modelled in the ontology, such as geography, dates and names, are simply ignored and not used to filter the results. There may also be some concepts not found in the ontology yet, or domain-specific terms that are not found in WordNet. In all these cases, adding keyword search may give the system the ability to make the results more relevant, by taking into account the words the user chose. Though a motivation of this system is to alleviate the need to use the correct words, a hybrid approach would give us the best of both worlds by

potentially getting datasets that are more relevant. Facets are also useful, allowing for further filtering.

- W6. **Hard to tag datasets:** Along with automatic tagging, the manual approach could be more streamlined to help the process go faster. It could also use automatic tagging to suggest concepts for datasets, making for a semi-automatic process.
- W7. **Imprecise search:** Compared to a regular keyword search, this approach helps find entries related to the words in the search query, adding datasets which may not have been found with the keyword search. Though this increases recall, the precision of the query is not improved, partly since the search query is not disambiguated. An effort to increase precision could mean asking the user for clarification (“did you mean..?”) or using a document collection to find the term the user most likely meant, using the terms’ context.
- W8. **Dataset contents not used for automatic tagging:** Though suggested as a potential improvement, the datasets themselves are not used to map datasets to concepts.
- W9. **Limited expressiveness in ontology:** The version 1 of the ontology only uses hierarchical “is-a” properties, meaning that concepts are only considered related if they reside close to each other in the resulting tree. This is not always right. “tollboth,” for instance, is considered more relevant to “garage” than “law,” since the former is a physical place (like tollboth), while the latter is an abstract concept, thus belonging to an entirely different branch in the ontology. Some “related” links across the tree could help combat this problem, though the search algorithm would need to be modified to support this.
- W10. **Standard semantic technologies not used:** DataOntoSearch version 1 does not use SPARQL, and rolls its own solutions instead. This makes it harder to compare the tool to semantic search engines that generate SPARQL queries, though it does enable ranking by how related the word is to the concepts.

Google Dataset Search is, of course, a real alternative to DataOntoSearch version 1. As explained in Section 3.1.1, Google’s solution is powered by Google’s search technology, and is designed to be a generic tool for finding datasets from any domain. However, just like with CKAN, the user’s query needs to match closely with the dataset metadata in order for the dataset to be retrieved. Some of the same problems evident with CKAN’s search are therefore present here as well. It should be noted, though, that Google Dataset Search solves the problem of finding the right open data portal, which previously required you to navigate to the right government body, which may not be obvious due to the way public services are split up between the federal government, states, counties, cities and municipalities, to take the United States of America as an example. That said, many countries host federated CKAN instances, like Data.gov, which collect datasets from all the different publishers and make them available in one place.

4.2 Research Questions

It is evident that there is a discoverability problem for datasets, and none of the systems available today have truly fixed it. Though DataOntoSearch version 1 is a start, it is a prototype, its suitability as a search engine has not been decided and a number of weaknesses have been identified. An eventual upgraded version could address many of the existing problems and hopefully bring better results. But to really bring improvements, we investigate how the first version performs, and what aspects of it can be improved.

My research question can therefore be split into two:

RQ1: What do users think of DataOntoSearch version 1? What main problems are there?

This addresses W1 above, and addresses W2 as a prerequisite.

RQ2: How can we address the identified problems when creating version 2?

This may additionally address the other identified weaknesses W3-W10 above.

4.3 Overview

The overall plan for the system mostly follows the two research questions, since the second naturally comes after the first:

1. Make the system ready for the pre-study.
2. Conduct the pre-study to find out what to improve.
3. Make improvements to the system.
4. Evaluate the newly improved system.

In practice, the amount of effort required for step 1 was much greater than anticipated, and led to a delay of the pre-study. There is also a significant amount of work involved in learning about the technologies involved and how such systems can be evaluated. Here is an overview over the project's timeline, starting from August 2018 and ending in June 2019:

August-September Try to understand the previous thesis and related background

October Research and design pre-study

October-January Understand and refactor existing system

January-February Continue work on preparing pre-study

February Conduct usability tests

March Set up system-oriented evaluation and analyze pre-study

March-April Integrate with CKAN

May Improve search

May-June Run evaluations and finish the thesis

The next two chapters go through the two research questions, one at a time. Chapter 5 tackles the pre-study, which constitutes the first research question. The planning of the pre-study, the work required to prepare DataOntoSearch version 1 and the results are presented in that chapter. Chapter 6 picks up right after, and presents the work that went into improving and evaluating DataOntoSearch version 2.

Investigation and results of RQ1

Our first research question is all about understanding where we are with version 1, and what direction we can take for further development. Though some simple queries have been tested using precision and recall in Hagelien’s master thesis and paper, we want to put the system in front of people to essentially get a reality check. This chapter goes through this process, starting with the test plan and proceeding with the implementation work involved and results.

5.1 Methodology

By the end of Hagelien’s project (see Section 2.4), we were left with a prototype for an ontology-based semantic search which had been evaluated against the CKAN search using information retrieval methods. Though it did compare favorably in some regards, there are some uncertainties related to its evaluation. I also wish to gain more insights into what should be improved, based on the one metric that actually counts – the end users’ problems with the search. A new competitor had also entered the stage since Hagelien delivered his thesis, namely the Google Dataset Search.

To achieve this, the pre-study is a competitive usability test.

I present the test plan here, using the questions described in *Usability Engineering* [34, p. 170] (summarized in section 3.4.2). In addition to those, questions of privacy considerations and the test methodology are included.

5.1.1 Goals

The pre-study aims to answer the following questions:

- How good is the current system, using Google Dataset Search as a point of reference?
- What should we improve? What does Google Dataset Search do better or worse than DataOntoSearch?

- What queries should we use in system-oriented evaluations later?

5.1.2 Test methodology

Here, some important questions regarding the test design are discussed. The questions discussed here do not have a particular connection to any of the other questions included as a part of the test plan.

Formative or summative? As explained in Section 3.4, there are two types of evaluations: Formative and summative. To achieve the goals of finding improvements, a **formative** evaluation seems most fruitful.

Competitive and choice of systems To evaluate against Google Dataset Search, the usability test is a competitive one, focusing on Hagelien’s prototype and Google Dataset Search.

Between-subjects or within-subjects? Competitive usability studies can be between-subjects or within-subjects. In order to achieve a reliable result, within-subjects testing is chosen.

Should the interface vary? Do we wish to find usability problems and differences in the interface itself, or do we wish to only test the search backends? If the answer is the interface, then we simply test with the different search interfaces directly. This also requires less work, and respects the fact that different search backends may be better suited with different frontends, for example due to autocomplete.

However, if we wish to test just the search backend, we need to use the very same user interface for all systems, with the least amount of changes needed to support the different ways of formulating queries. This will potentially require a lot of work, since Google Dataset Search doesn’t support APIs.

Though it is desirable to use same interface for all, it is not essential. Due to the amount of work required to create a unified interface, we must make due with using different interfaces.

Should the available datasets vary? In the beginning, the only datasets with a manual tagging in Hagelien’s system were datasets fetched from different Norwegian sources. These datasets are not available in Google Dataset Search. We therefore have a couple alternate approaches:

1. Expand what datasets are available in Google. This cannot be done, since we are not in charge of DIFI or Google.
2. Expand what datasets are available in DataOntoSearch. Though this can be done, some effort must be spent manually linking datasets to concepts.
3. Test using different datasets for different systems.

Approach 3 above is not useful at all. Using different datasets for different systems means information retrieval methods are out of the picture, since differences in performance might be due to different datasets. Of course, this also applies to the usability test – we do not know if it is the interface/search engine or the available datasets that makes the difference. An additional problem is the fact that different datasets means using different tasks, which further reduces what the pre-study can tell us about the systems under test.

Since approach 1 above is out of our reach, we are left with approach 2, i.e. add datasets to DataOntoSearch that are indexed by Google Dataset Search. The work that has gone into this is described in Section 5.2.2 on page 50.

5.1.3 Collection and analysis of data

Collecting personal data would entail satisfying a long list of requirements to ensure the test users' privacy is handled properly. Perhaps the greatest hurdle is the process of sending a form to the NSD¹, which evaluates and makes sure the planned research fulfils our privacy duties. Since they need some time to look at the application, it must be sent in a couple weeks in advance. I was not made aware of this before it was time to conduct the pre-study, which means the study had to be designed so that no personal data is collected. That way, our study will not infringe the test users' privacy, and will not require any application to be sent.

With this in mind, the following data will be collected:

- Recording of screen
- Search queries
- Hand-written notes focusing on user comments and struggles
- SUS score
- Comparative questionnaire
- Answers to the tasks

Using these, we will arrive at the following measures:

- User's accuracy
- User's efficiency with the system
- Pain points and other interesting findings
- How satisfied was the user with each system?
- Which system was preferred by the user? Why? By how much?
- Collect issued queries

¹<https://nsd.no/english/>

The recording of the screen will be there for later reference, so we easier can judge the system's performance. It will also let us see how fast the user was. It will be used on an on-demand basis. The recording must be started and stopped by the experimenter, and saved to the local disk. After the test sessions, the recording should be moved to a permanent storage, like a network folder at NTNU. A random identifier should be created for each test user, so the recording and notes can be associated with each other.

Searches performed by the users should be analyzed to reveal search queries that users expected to give valuable results, but which the system failed to deliver on. They will be collected by inspecting system logs, or by looking through the screen recording.

The hand-written notes will be the primary source of analysis. User comments should be transcribed to a more usable medium shortly after the test session, with some extra details filled in from memory. They should be analyzed to find problems with the system that the users struggled with.

The SUS score will be collected after the user is done with one system, before moving on to the next. The standard questionnaire will be given.

The comparative questionnaire is given after the final system's SUS questionnaire has been delivered. It will ask questions about what system the test user liked the best, and why. It should also ask the user to rate the other system, in comparison to the "best" system, again asking for their reasoning. This should supplement the information derived from the notes, and also give us an idea of how well the current system fares compared to Google.

5.1.4 Test details

The remaining topics from the test plan are presented below.

When and where to perform the study

The tests will be performed in a bookable group room at NTNU, or another location depending on the availability of rooms.

The time must be decided depending on when the participants are available.

Duration

We aimed at using no more than an hour, with the ideal time being around 45 minutes.

Computer support

The main need is for screen capture, in addition to the capability to run the software under test, of course. To avoid problems with the untraditional trackpad on my computer, an external mouse should be used.

Software to ready

A web browser must be set up with the systems to test. The backends must be running (for any system that I set up myself), and it must be ready for use. Two bookmarks should be

provided in a bookmark bar, so the user can easily navigate to the different search engines. The browser should have one tab open, namely the first screen of the first system to test.

Software for recording the screen must be armed for recording.

System/network load and response times

We use DataOntoSearch running locally. This means its results are not delayed by the network as much as Google is. Since the system is not very quick and takes a couple seconds to retrieve documents, we opted to not delay or throttle the traffic. The performance has not been a focus, and it is therefore not interesting to test since we already know it's poor.

Experimenters

I will be the sole experimenter. Since we're only testing acquaintances of mine, it is not as important for me to have another experimenter as in cases where we test using strangers.

Who are the test users? How to recruit them?

For this formative usability study, I and my supervisors decided that it is okay to primarily recruit acquaintances of mine. This helps me recruit users fast enough to conduct tests in a timely manner. It also avoids the problem of registering personal details through electronic solutions.

How many test users?

4-5 test users is a good enough amount, since we only test two systems.

Test tasks

The tasks must be related to the datasets used, and therefore limited to the transportation domain. They should focus on describing a need for the user, which they must use the search software to solve.

It is important that the name of the datasets and other features of the datasets aren't revealed in the task descriptions. Users may be led by them, and have their thought process swayed (primed) by the formulation of the task.

Another aspect to consider is whether the user can assume there is a relevant dataset. If they can, then the situation becomes unrealistic, since they may then have the tenacity to continue searching, long after the point where they would normally have given up. On the other hand, if the user doesn't know whether there are relevant results or not, they might give up easily. And users who give up on the entire test may be "correct" some of the time. To make it more realistic and hopefully learn something new about how users judge the fitness of a certain search engine, the latter approach is the preferred one.

Please refer to appendix A.2 for the tasks that were given to users.

Criteria for finishing test task correctly

To have finished a test task, the user must do one of two things:

1. Declare that there is no dataset fit for the task
2. Declare one dataset as the one fit for the task

For this to be a correct answer, the answer must coincide with what actually is the case, depending on the task.

To make it easier to see in the screen recording, we might want to use some software to signal the answer, rather than the user simply telling the experiment runner what they think.

A concern with this method is that it does resemble a test of the user's ability to find an answer, since there is a right and wrong answer. As such, they might feel intimidated and wish to succeed. This must be compensated for by our introduction and guidance, so we don't stress the user.

User aids

The user will have access to a short manual which is printed out, given alongside the instructions. The manual should describe the characteristics of the systems, giving some guidance on how to effectively use them.

See Appendix A.3 for the concrete manual given.

In addition, users may ask questions regarding name of relevant companies, geography, translations and other questions that they would normally have found answers to using aids like Google or Wikipedia.

Help from experimenter

The experimenter will not help the user with formulating the query or picking a dataset. However, if the user has any specific question about the task itself and its context, like the name of the transport company in Stavanger, the experimenter may answer as much as they think is necessary. The aim is to ensure users have an equal chance of formulating relevant queries, independent of their knowledge of the geographical locations used in the task.

The experimenter should ask the user what they would do, had the experimenter not been available. The experimenter replaces typical Google and Wikipedia usage.

Criterion for announcing the interface a success

This point is not so relevant for this formative study, but it is included for completeness. A successful interface:

- Should be pleasant to use. We measure this as having a good SUS score. This is difficult to quantify since SUS is not absolute, but it should be above average.
- No major problems should be found.

- Users should be able to formulate successful queries with no more than two attempts.
- Users should be able to find a fitting dataset within one and a half minute.

Budgetary concerns

There are no extra costs associated with the pre-study.

5.2 Implementation

Before I can evaluate DataOntoSearch version 1, there are a couple prerequisites that must be fulfilled:

1. I must have a running version of the system.
2. A set of datasets which are available to DataOntoSearch and Google Dataset Search must be found.
3. The system must be populated with the set of datasets I found.
4. Manual associations between datasets and concepts must be added.
5. The ontology must be modified to accommodate the new datasets.

This turned out to be a challenge in and of itself.

5.2.1 Getting a runnable system

The system that was used as a starting point for this master thesis was delivered by Thomas Hagelien at the end of his own master thesis project. It showed signs of being a snapshot at a point in time, which could result from an experimental approach which suddenly had to halt so it could be delivered. Specifically, the software system had the following problems:

- **Hard to read:** Abbreviations were frequently used. To make matters worse, meaningless variable and function names, like “foo” and “x,” were found frequently. This made it hard to understand the software.
- **Poor organization:** The software was very difficult to modify, since the relationship between modules was not very clear, and large pieces of code were duplicated across the places it was used. A lot of technical debt had accumulated and needed to be taken care of in order to restore modifiability. There was also a lot of failing code that was not referenced anywhere, which made it difficult to know what code was important and what was leftovers from earlier phases.
- **No database setup:** There only existed code that used the database, and no code for adding graphs to it. Parts of this was available in Hagelien’s master thesis, although parts of the code was cut off since it went outside of the page boundaries. This code was only made available when I asked Hagelien about it, and even then, the code was only available as Jupyter notebooks, which essentially are printouts of interactive code sessions.
- **Code that fails:** Certain functions try to call functions that do not exist. This is present both in central classes and in code used for evaluating results. It also looks like there was a change in how graphs were organized at one point, since some parts of the code refers to an outdated way of organizing graphs. Specifically, the dataset tagger did not work since it added similarity links to a graph, assuming the very same graph should be used for ontologies, datasets and manual taggings. These errors and their nature were not marked in any way, and had to be discovered manually.

- **No instructions for how to run:** There were no instructions for what you had to do in order to run the system, as is typically found in README files. This, too, had to be learnt by way of email correspondence with Hagelien.
- **Hard coded MongoDB instance and graphs:** Only the MongoDB database login was set to be configured by the user. The rest had a value set in variables in the code, requiring code to be edited in order to change the MongoDB instance used or the graphs used for various purposes.
- **Hard to run:** Only the webservers for dataset tagging and searching existed as scripts you could run. The rest of the system was only available as Jupyter notebook files, as previously mentioned, which cannot be run as easily.
- **No version history:** The finished project files were added in huge commits to the Git repository around the time of the delivery deadline. The commit messages gave no additional information about the intentions behind the system. It seems like the public Git repository simply functioned as a way of hosting the final project, and was not used during development.

Essentially, it would not have been possible for someone else to replicate the master thesis as it stood. They would need access to files that were unavailable, and would have no instructions for how to get the system running. Had there not been for Hagelien's support, I would have been unable to get the system to run and would have had to resort to recreating the system from scratch, barely achieving anything new with the thesis.

In order to get the system into a state where further development could take place, my first work on the system was refactoring work. Essentially, my aim was that if someone else wants to run the system, they should be able to do so without much trouble. This work also doubled as a way of getting familiar with the system and its design, so I could get a better grip of how to develop it further.

The refactoring work took a couple months, and was an on-going project since new parts of the system were needed later on in the project. For example, the evaluation code had to be completely rewritten when a need for system-oriented evaluation arose. After this, I dedicated some time to keeping the system in a modifiable state, so it would keep the desired qualities when I delivered my master thesis.

The work resulted in what you could call DataOntoSearch version 1.1, though I will simply refer to it as version 1. Since I wanted to evaluate the system developed by Hagelien, I made an effort to keep the search logic functionally equal to the existing system, so that any findings relate to the existing system, and not something I wrote.

Specifically, the following improvements were made:

- **Create README:** I created a document which describes the background of DataOntoSearch, and how to set up the project to run locally. It is included in Appendix C.1.
- **Use environment variables:** Instead of hardcoding what graphs to use and which MongoDB instance to connect to in the code, environment variables are used to set these parameters. They provide a flexible and universal way of specifying settings that change between different deployments of DataOntoSearch. In addition, so-called `dotenv` files supported, so the environment variables can be put into a file instead of being supplied through the shell.

- **Use one entrypoint:** Instead of having many different Python scripts laying around and leaving it to the user to find the proper script to run for a given task, the user only uses a single script. This script functions much like `manage.py` in Django and command line programs like `git`, in the sense that you use different subcommands to do different tasks in the system, with built-in help functionality which lets you discover available subcommands and arguments.
- **Centralize repeated code:** Database interactions and all other kinds of repeated code is put into a central place, and re-used wherever they are needed. The process of choosing what to include there and what to keep in specialized files, were mostly done after-the-fact, instead of trying to anticipate what would be useful abstractions. Some duplication is still present, since there is a trade-off between decreasing duplication and increasing complexity. For example, the code for handling different types of graphs is re-used, but for a different type of entity called Configuration, different code is used.
- **Create subcommands for handling database entities:** The so-called CRUD tasks (Create, Read, Update, and Delete) are all implemented as separate subsubcommands for the different subcommands that correspond with different types of database entity (ontology, dataset, similarity, autotag and configuration). For example, running `python dataontosearch.py ontology create` creates a new ontology graph in the database, using the ontology created as a part of this project.
- **Create subcommands for searching from command line:** Instead of only performing searching through the web interface, separate subcommands are available that lets you automate searching to a greater degree. This is also utilized for an automatic evaluation utility.
- **Remove unused code:** After checking what code is used and not, functions and files that were not in use were removed. This way, it is easier for newcomers to know what to pay attention to.
- **Make code more readable:** This work involved renaming variables, creating new functions, adding explanatory comments, storing data in mapping-like objects instead of multiple arrays and restructuring the project.

For this last point, compare the code that handles the search procedure, shown in Appendix B on page 125.

5.2.2 Finding common datasets

When we want to compare different systems in a competitive usability test, we want to reduce the number of variables between the systems. That way, any differences we observe are more likely to be caused by the systems themselves, and not irrelevant variables.

The datasets available to the systems is one such variable we wish to eliminate. Though DataOntoSearch is capable of importing any datasets that are expressed with DCAT in RDF, the system is dependent on associations between datasets and concepts. Settling

on using the autotagger would likely mean decreasing the quality of the search, since the autotagger is far from perfect, especially when faced with incomplete dataset descriptions.

The set of datasets we wish to find, must fulfil the following criteria:

1. They must be indexed by Google.
2. They must be possible to import into DataOntoSearch.
3. They must be at least a little related to the ontology we use, in our case the Open Transport Data ontology.
4. The number of datasets must be manageable, so a manual tagging process can be performed.
5. The datasets must be relatable to the test users.

After some experimentation and looking around, I ultimately landed on using datasets from the State of New York that are put in the transportation category. There are 288 datasets that match this criteria. These datasets are indexed by Google, and are available from the American federated CKAN instance² as well as through the system ran by the State of New York themselves³. The latter system hosts both datasets from New York State and federated datasets from Buffalo City and New York City, among others. In addition, New York is a famous location, and although not so many Norwegian students are familiar with its geography or details, I assume most people can imagine the city based on what they have seen in popular culture. They may not be so familiar with the *state* of New York, though.

To get those datasets into DataOntoSearch, they were imported from the State of New York's Socrata instance into a local CKAN instance that was running on my computer. From there, it was imported using the existing scripts for importing datasets from CKAN, limiting the datasets to those with the "transportation" tag. The resulting RDF is available in the code repository⁴ and can be imported directly into DataOntoSearch.

A set of associations between those datasets and our concepts – the so-called manual linking – has been created by Dr. Shanshan Jiang and Marit Natvig, so that the datasets can be used with DataOntoSearch without relying on the automatic tagging. This is also accessible in the code repository⁵.

5.2.3 Updated ontology

The ontology is an important part of the DataOntoSearch project. Its structure and concepts determine how well the system can represent the user's query, and how well it can represent all the datasets. It also affects how well the system is able to show datasets that the user did not mention explicitly, but which are still related to their query.

²<http://catalog.data.gov>

³<http://data.ny.gov>

⁴https://github.com/tobinus/OTD-semantic-framework/blob/master/doc/ny_dataset.ttl

⁵<https://github.com/tobinus/OTD-semantic-framework/blob/master/doc/manually-tagged-ny-datasets.xlsx>

The types of datasets found in Section 5.2.2 are a little different from those used with the ontology before, so the ontology was updated to account for this. Some concepts were also moved, to make the new hierarchy more logical. This was done based on the input from Jiang and Natvig.

Before the refactoring, the ontology was defined using Python code which generated the corresponding RDF graph. As a part of the refactoring, this was changed so that the ontology is stored in Turtle format, an RDF storage format which is convenient for humans to read and write. The concepts are sorted in depth-first order so the hierarchical structure is evident. This file is stored in the GitHub repository ⁶.

In addition, I created a tool for generating HTML visualizing the concept hierarchy. The tool helped the domain experts, Jiang and Natvig, when they were performing the manual linking of the datasets.

5.2.4 User interface improvements

Some small changes were done to the user interface to make it more presentable to users:

- Concepts that matched the query and datasets are hidden by default, and can be shown by checking a checkbox. You can see this in action in Figure 5.1 and 5.2.
- The search query textbox has concept labels as search suggestions.
- Some smaller stylistic changes, including using white-space to separate the individual results from each other.

The following improvements were made after the pilot test:

- The dataset links point directly to the original publication, instead of pointing to the CKAN instance they were imported from.
- The choice of similarity type was temporarily removed to avoid confusion.

⁶<https://github.com/tobinus/OTD-semantic-framework/blob/master/ontology/otd.ttl>

DataOntoSearch

Using synonyms and near-by concepts to fetch (hopefully) relevant datasets.

Query

Show matching concepts

Matching datasets

[For Hire Vehicles \(FHV\) - Active](#)

TLC authorized For-Hire vehicles that are active. This list is accurate to the date and time represented in the Last Date the contents of this dataset, please email licensinginquiries@tlc.nyc.gov.

Score: 0.972

[Freight Cars Moved in Carfloat Operations, Port Authority of NY NJ: Beginning 2009](#)

This dataset provides volume of loaded freight rail cars transported between NY and NJ by New York New Jersey Rail NJ. Total volume, including empty cars, is estimated at twice the volume of loaded. 2013 entry is year-to-date volume and

Score: 0.969

Figure 5.1: The revised web search interface, without concepts being shown.

DataOntoSearch

Using synonyms and near-by concepts to fetch (hopefully) relevant datasets.

Query Tagged

Show matching concepts

Concepts derived from query

bicycle	1.0
motorcycle	1.0
car	0.917
carsharing	0.917
vehicle	0.889

Matching datasets

[For Hire Vehicles \(FHV\) - Active](#)

TLC authorized For-Hire vehicles that are active. This list is accurate to the date and time represented in the Last Date Updated and Last Time the contents of this dataset, please email licensinginquiries@tlc.nyc.gov.

Score: 0.972

rental	1.0
vehicle	1.0
license	1.0
motorcycle	0.889
train	0.889

Figure 5.2: The revised web search interface, this time with concepts.

5.3 Results

The results of the pre-study are presented here, including the answers to the questionnaires and task performance.

There were four test users (N=4). They all attend study programs closely related to computer science, and can be considered developers. They were all acquaintances of mine or friends.

5.3.1 Changes made from pilot study

The first usability test conducted served as a guide on what to improve for later tests. I don't expect the problems to have much of an effect on the systems' performance, but they mostly concern the test itself. The problems discovered, along with the changes made to remedy them, are presented below.

- The delivery form, based on Microsoft Forms, was too difficult to use. Particularly because it was impossible to split between pages, and some fields were not shown before a selection was made.
 - I re-created the form in Google Forms and made it easier to use.
- The user was not aware that I could answer some questions.
 - I added this to my test introduction.
- The test went on for too long, and the tasks were easy when tested on the second system, since the user knew what dataset to look for.
 - The remaining tests were done with two tasks per system instead of three.
 - One random task is made exclusive to the second system.
- The user got confused about the meaning of “tagged” versus “auto,” which were presented in a selection field to let the user choose between the manual concept associations and the autotagger's associations.
 - The select box letting the user pick between “tagged” and “auto” was removed. Instead, “tagged” is always chosen.
 - I do not consider this an essential part of DataOntoSearch's operation, but rather a by-product from the prototype phase. It therefore does not need to be a part of the evaluation.
- The user was not so familiar with how to think aloud.
 - I added a demonstration of thinking aloud to my test introduction.
- The user was concerned about finding datasets that were up-to-date.
 - The test tasks were reworded so that their timely aspect is no longer emphasized.

- The user encountered many datasets from outside New York in Google Dataset Search, since Google is not restricted to New York datasets.
 - I added a note to myself that the user should be made aware of the “site:” operator available in Google Dataset Search, along with potential sites to use it with.
- The user was confused about how to copy and paste on my computer.
 - I added a note in my test introduction to tell the user that my “Ctrl” and “Fn” keys have switched behaviour with each other.

During the second user test, two smaller problems were discovered and fixed:

- “DataOntoSearch” was mistakenly named “OntoDataSearch” a couple places.
 - I updated the documents so “DataOntoSearch” is used consistently.
- The user thought task 68 asked for the existence of train routes, not specific train departure times.
 - Task 68 was reworded so it specifically asked for train departures.

Even though there were some problems, I do not think their presence biased the test in favor of any one system. The results from these tests were therefore kept and are presented along with the results from the other two tests in the following.

5.3.2 User rating of systems

The ratings done by the users, randomly labelled A, B, C and D, are presented in the following table. They were collected through the comparative questionnaire, using a scale from 1 to 9.

	Very bad					Very good			
	1	2	3	4	5	6	7	8	9
Google Dataset Search						ABD	C		
DataOntoSearch			AB	D	C				

All users rated Google Dataset Search above DataOntoSearch version 1. Two users had a three point difference between the two systems, while the other two had a two point difference.

The average rating in general is 5, which is right in the middle of the scale. Google’s average rating is 6.25, while DataOntoSearch’s average rating is 3.75.

User	Google	DataOntoSearch
A	60.0	67.5
B	60.0	65.0
C	70.0	50.0
D	70.0	57.5

Table 5.1: SUS scores for Google Dataset Search and DataOntoSearch version 1. The bold numbers indicate that the system was the first the user tested.

5.3.3 User satisfaction

SUS scores are calculated from the collected SUS questionnaires. Users were consistently more satisfied with the second system they tested, independent of whether this was Google Dataset Search or DataOntoSearch. That said, the increase was more pronounced for users who went from DataOntoSearch to Google Dataset Search. Though note that this kind of comparison is not reliable, due to the time gap between the questionnaires and the lack of a frame of reference, as noted in section 3.4.2. The raw scores are reported in Table 5.1. The average SUS scores were 65 for Google Dataset Search, and 60 for DataOntoSearch.

5.3.4 Problems and themes in user comments

User comments were collected directly from users in the comparative questionnaire, where they were asked to explain their rating of the two systems. In addition, some comments were collected as written notes by the experiment runner while the user was trying out the system. Here, common problems and other comments for DataOntoSearch are presented.

- **The system failed:** Users could not find what they were looking for, and always answered that the dataset did not exist. One user reported that “I did not get any desired results, as far as I could see.” Another one wrote that “Didn’t find any database needed.”
- **A few datasets are always on top:** Especially the “Bus Safety Network” dataset is always among the first, even if you try to vary your query or search for trains. As one user put it, “Always gave same (wrong) results.” Another user attributed this deficiency to the ontology approach, noting that “Ontology method ‘works,’ but search results are too similar if used to the keyword approach, and a bus database nearly always got top score.”
- **Unprofessional look:** The look, though simple, did not evoke the same kind of authority as Google’s professional design, with users noting that it “Looked very simple” and “A bit boring UI.”
- **Matching concepts were appreciated:** Users felt more in control of the system when they could see concepts that the query was matched with. They also liked how they didn’t need to try different synonyms for words in their query in DataOntoSearch. Users noted that they “Liked the word comparison, but it was not clear how the words were matched” and “Liked the check box with synonym for my query.” Not all users discovered this functionality by themselves.

Measure	Google	DataOntoSearch
Average comparative rating	6.25/9.00	3.75/9.00
Average SUS score	65/100	60/100
Success rate (tasks with dataset)	3/5	0/6
Success rate (task without dataset)	3/4	3/3
Success rate (total)	6/9	3/9
Median task duration	00:04:53	00:05:45

Table 5.2: Summary of the measurements collected from users when testing Google Dataset Search and DataOntoSearch version 1.

5.3.5 User performance

First, we look at what the users answered. For Google Dataset Search, there were 9 attempts at a task. Six of these were successful, while the datasets chosen for the remaining three would not have helped the user solve their task and therefore registered as incorrect. Two of the incorrect answers may possibly be explained by test users not understanding exactly what the task was asking for, though this can also be attributed to how the dataset is presented in the search result page. For all tasks for which a dataset exists, the users answered that there existed a dataset. Google Dataset Search was thus able to provide search results which users felt were right for the task.

For DataOntoSearch, on the other hand, not a single user answered that they had found a fitting dataset. For one task, this was the correct answer, which means that the users that tested this task with DataOntoSearch found the correct answer. These are the only correct answers for DataOntoSearch. For all six attempts at solving the other tasks, the user did not find a fitting dataset, even though it should have been available. Thus, for all the datasets that were returned, the user deemed them not relevant enough.

The median duration for task attempts on Google Dataset Search is 00:04:53, while the median duration for DataOntoSearch is 00:05:45.

All of the measurements are summarized in Table 5.2. Note that the low number of users means these numbers cannot be said to represent all developers in general, but they give us a better indication than testing with no users, and they do tell us how the systems fared in the hands of the recruited test users.

Implementation and results of RQ2

Equipped with a better understanding of the issues users faced using DataOntoSearch version 1, the mission to improve DataOntoSearch can begin. Of course, just tweaking things and hoping an improvement was made will not do. I will need a sound way of assessing the improvements. This chapter describes how the evaluation is planned, the implementation changes made and how the changes affected how well the system performs.

6.1 Methodology

After the system has been developed further, we need to know whether we actually made any improvements, and by how much. We also want to know how the system compares to Google Dataset Search. This section presents the methodology used for the final evaluation.

6.1.1 Goals

The overall goal is to know what this thesis contributes to the field, and provide evidence of this contribution. Specifically, I must learn:

- **How did we do?** Were the improvements a step in the right direction? Did we see only small benefits, or was it a bigger shift? Do the changes apply universally, or only to certain types of queries? This involves comparing with an earlier version of DataOntoSearch.
- **How do we compare to the alternatives?** Is DataOntoSearch a viable option, when set up against Google Dataset Search?
- **How does the system react to different threshold values?** There are a couple variables that can be varied to tune the search system. It would be nice to know what value they should be given, so potential users of the system have a good starting

point. It would also be interesting to see how much they vary between different queries, and how important it is to select the right threshold values.

- **How fast does the system run?** Though not a primary focus, it is interesting to know how much time the search procedure takes.

In contrast to the formative nature of the pre-study, whose purpose was to give a direction for further development, this new evaluation is summative. It must give us some numbers that tell us something about how well the systems fulfil their purpose.

6.1.2 Measures

Whereas comments from a few users give us a valuable insight into their needs and can inform our direction, they are not reliable enough to conclude what system is best, at least not on the scale of the pre-study. It is also necessary to test with many variations of the system, which would be difficult to achieve with the overhead and effort required by a usability test. I'm therefore using system-oriented evaluation methods for this evaluation. These methods were introduced in Section 3.3 on page 23.

Specifically, the following evaluation methods are used:

- **Precision:** Tells us to what degree irrelevant datasets pollute the results.
- **Recall:** Tells us whether the user can find what they're looking for, given their query.
- **F1 score:** Gives us a unified measure of precision and recall, reconciling the two.
- **R-Precision:** Incorporates an aspect of ranking to precision, by looking at only a top portion of the retrieved datasets.
- **Mean Average Precision:** For properly distinguishing systems that rank relevant datasets high from systems that rank them low.

For all these measures, I must arrive at a set of queries and a set of relevant datasets for each query in some way. Since it is easier to use binary relevance assessments rather than graded ones, only measures made for such relevance assessments are used.

In addition, there is one goal related to the system's time usage. For this purpose, the execution time serves as a good enough measure, though care should be taken to eliminate external factors like different workloads on the computer.

6.1.3 Queries and relevance assessments

As I explained in Section 3.3, the choice of queries greatly affects the evaluation's validity. If the selection of queries does not properly reflect the type of queries users would make, we end up with numbers that don't tell us anything about how the system would perform "in the wild," with actual users. There is also the danger of "overfitting" the system, which is the trap of over-optimizing the system for one set of queries. You may then risk ending up with a system that performs poorly once you try out a query it has not been optimized for.

Concept-based relevance assessments

There is a significant amount of work required to manually go through the available datasets and mark them as either relevant to the query or irrelevant. We may therefore want to take a shortcut. We already have assigned concepts to each dataset manually, to enable the operation of DataOntoSearch. What if the assignment of datasets to concepts can also be used to determine each dataset's relevance to a query?

Usually, the relevance assessments are done once the queries have been decided upon and we know what datasets were retrieved. If we use the concepts, however, we must go the other way around:

1. Find a concept which is associated with several datasets
2. Formulate a query so that a dataset is relevant to the query if and only if it is tagged with the related concept

There is a rather obvious problem with this approach: You are essentially using one extra piece of information, namely the manually tagged datasets, for both DataOntoSearch's search procedure and for the system evaluation. This is information which e.g. Google Dataset Search does not have access to. These queries may therefore risk favouring DataOntoSearch over other search engines.

To combat this, some extra care must be taken when formulating the queries. If you directly quote the concept label, the DataOntoSearch search procedure is more or less guaranteed to match with that concept. Using alternate ways of describing the same concept should alleviate this effect. That said, search interfaces using DataOntoSearch may choose to use concept labels as search suggestions, which one would assume makes such queries more representative of the type of queries users would make. But then again, Google Dataset Search uses full dataset titles as search suggestions, which would mean using a dataset's full title might represent users of Google Dataset Search better. For the purposes of this thesis, I will simply acknowledge this problem, and ensure anyone interested may look at the queries and concepts to determine the validity of the results.

There are three sources used to gather queries:

- Queries derived from concepts in Hagelien's thesis.
- New queries derived from concepts by me.
- Queries collected from the usability tests, with concepts found afterwards.

Thomas Hagelien already used this approach in his master thesis [20]. The queries he made were made with another collection of datasets in mind, namely datasets related to transport in Norwegian. Some of the queries made for that dataset collection are associated with concepts that are not used in the manual tagging of New York datasets, and therefore had to be dropped. The remaining four queries are re-used.

To increase the number of queries, I found more concepts and made queries for them. My approach was:

1. Choose one concept at random, using a random number generator.

Table 6.1: Queries with concept-based relevance assessment

Source	Query	Concepts
Hagelien’s thesis [20]	location ¹	Location
	statistics ¹	Statistics
	bike	Bicycle
	map	Map
New, from concepts	messaging	Communication
	stopping place	Station
	walking	Pedestrian
	driving permit	Driver’s license
	tallying	Counting
	road traffic management	Traffic regulation
Usability tests	new york state rest stop	Rest area
	nyc subway	Underground railway
	rest area	Rest area
	site:data.cityofnewyork.us subway stations	Underground railway \cap Station
	subway stations	Underground railway \cap Station
	rest stops	Rest area ²
	highway new york rural	Highway ³
	underground railway posi- tion	Underground railway \cap Railway ³

¹ This query had to be modified by appending “transportation,” since Google reached its upper limit of datasets returned.

² This concept was chosen by me, not the domain expert.

³ There are no matching datasets, so this query was not used.

2. Check if there are any datasets associated with the chosen concept.
3. Use Wiktionary and Wikipedia to arrive at a term or phrase that has more or less the same meaning as the concept, but is lexicographically different from the concept’s labels.

I decided beforehand to do this until I had six more queries.

The final source of queries were the usability tests conducted as a part of the pre-study. The users’ queries for both systems were collected. For this purpose, I used the first query each user made, since I assume the best case would be for them to find the relevant dataset on their first try. Based on the query, our domain expert found concepts which should more or less match its intent. It was not possible to do this for all queries, since some combinations of concepts did not have any datasets associated with them.

The queries and the concepts they are derived from are listed in Table 6.1.

Task-based relevance assessment

Of course, one of the goals of the pre-study was to understand what type of queries users make when sat in front of a new, semantic search engine. For this purpose, queries made

by the users were collected. Since there was one dataset indicated as relevant per task, we could use queries the users made and say that only the dataset relevant for the related task is considered relevant for that query. This would effectively work like a simulation of the usability tests, letting us know whether the improved DataOntoSearch would fare better in those tests than version 1, at least for the search engine aspect.

Having just one relevant dataset is highly unusual, and makes it more important to look at the ranking of datasets – if the relevant dataset was returned first, the system should not really be punished for returning many irrelevant datasets afterwards (which decreases the precision and F1 score). Presumably, the user would not care about there being extra datasets after they found what they were looking for in this context. Indeed, one of the test users noted that:

When I search, then I may read, like, the first five hits, and then skim through the rest. But if I don't find what I'm looking for among the first five hits, I can almost be certain that I won't find it at all.

The Mean Average Precision (MAP) measure neatly captures the search engine's ability to rank the relevant results high.

As for how to pick the queries to use out of all the queries the users made, we can use a procedure like the following:

1. Focus on the tasks for which a relevant dataset existed. See Appendix A for the task descriptions.
2. Look at one user and one task at a time.
3. Given the list of queries the user made, we throw away any “testing” queries, like “hello.”
4. Users often made multiple queries. However, I assume that the best case scenario is the user finding what they look for with their first query. Therefore, we should pick only the first query and add to our list of queries. The queries the user made after the first one are likely influenced by the results from the earlier queries anyway.

Since users tested both Google Dataset Search and DataOntoSearch, we could potentially use different queries for the two systems. Since we want to do a direct comparison between the two systems, we should not vary the queries between them. Instead, queries collected from both systems should be run on both systems.

The queries used with task-based relevance assessment are listed in Table 6.2, along with the task they were taken from and the system in use.

6.1.4 Collection and analysis of data

With the queries and their relevant datasets decided, they need to be run with the systems to see what results they retrieve and calculate the metrics described earlier.

Although it would be *possible* to use DataOntoSearch unmodified to run the queries and write down retrieved datasets, and perform the relevance assessments manually, this would take much effort. By adding the capability to run such evaluations automatically,

Query	Task	System
underground railway position	Task 55	DataOntoSearch
site:data.cityofnewyork.us subway stations	Task 55	DataOntoSearch
nyc subway	Task 55	DataOntoSearch
subway stations	Task 55	DataOntoSearch
highway new york rural	Task 47	DataOntoSearch
new york state rest stop	Task 47	DataOntoSearch
rest area	Task 47	DataOntoSearch
nyc subway station	Task 55	Google
rest stops	Task 47	Google
new york state highway public facilities	Task 47	Google
new york public facilities	Task 47	Google
new york road	Task 47	Google

Table 6.2: Queries derived from the pre-study

a greater number of queries can be used, and evaluations can be done more frequently to give us a better picture of how smaller changes affect the system’s performance, rather than run as one big batch at the end.

As part of the implementation work, I have implemented the capabilities needed for such automatic evaluations. They are split into two parts: One part reads a recipe on what queries to run, and what values should be used for the different variables. All combinations of value assignments and queries are then executed, using the cartesian product. The results are reported for each execution. The other part is fed this result, does the relevance assessment using the same recipe, makes the needed calculations and finally reports the results. For the current DataOntoSearch system, the two parts directly interact to run the queries and process the results.

Older version

The first version of DataOntoSearch also included some capabilities for automatic evaluation, but they reference functions that do not exist and were not available as simple scripts. However, as a part of the refactoring work, a command line utility had been created. It allows for running one query at a time, with machine-readable results. I am therefore able to manually run the queries, capture their results and reformat them so they are in the same format as the query results from the current DataOntoSearch system. This can then be fed into the same evaluation code as used for DataOntoSearch version 2.

Google Dataset Search

For Google Dataset Search, the query-running module has been extended with the option of using Google Dataset Search instead of DataOntoSearch. This way, the same evaluation code can still be used, since just the process of retrieving the results is different.

To gather results from Google, the software must look at the HTML returned by the search, and compare the dataset titles, since no public API exists. The collection of searchable datasets varies between the two systems, so you may specify a “site:” operator to

restrict results to one dataset provider. In addition, any unrecognized datasets are ignored, so the resulting list of datasets from Google answers the question “what would Google retrieve, if they were limited to the same dataset collection as DataOntoSearch?” Unrecognized datasets may still take up spots that could otherwise be occupied by recognized datasets, that may have been pushed off the search result page as a result. To avoid this problem, reports of unrecognized datasets should be checked to ensure we are not approaching the maximum number of datasets displayed in the search result page.

Doing the analysis

Evaluating the results once gathered is fairly simple. The recipe that lists queries, also lists datasets or concepts relevant to that query. The evaluation code may therefore look through the available datasets, find the relevant ones, and check what retrieved datasets were relevant and mark the rest irrelevant. The precision, recall, F1 score and MAP measurements can then be calculated, since we both have a set of relevant datasets in the dataset collection, and an ordered list of retrieved datasets and their relevance.

Time spent

Finally, the system’s execution time must be measured. To measure it, log messages with timestamps can be used to check the time between the start of a search and the moment results are retrieved. However, the reported time will also be affected by other tasks running on the computer, since the search process will share processing power with the other tasks.

A more robust time measurement tool, like the `perf stat` command-line command, can be used to run the program multiple times and report many interesting statistics like the time spent in the CPU, but it will only look at the software as whole. It cannot report which part of the search process took the most time. It should also be noted that a significant part of DataOntoSeach’s execution time is spent on I/O, which contributes a lot to the execution time. Looking at just the CPU runtime would not account for any improvements (or disimprovements) in the application’s I/O usage.

A combination is likely to yields the most useful information.

6.2 Implementation

A major part of this thesis' contribution lies in the improvements that are made to the implementation. Apart from the refactoring work described in relation to RQ1, the changes and new developments made to the implementation are presented in this section. The code of DataOntoSearch version 2 is published on GitHub¹.

6.2.1 CKAN Extension

One of the complaints from the usability tests, was that the design of DataOntoSearch's web interface did not look so professional. In addition, Dr. Shanshan Jiang noted that it would be difficult to make users use DataOntoSearch if it is a separate, third-party website which is not linked to from the dataset archives. This goes for both dataset consumers who would like to find a dataset for their needs, and dataset publishers, who are unlikely to go to a separate website to associate some concepts to their datasets.

To solve these problems, the functionality of DataOntoSearch has been integrated with CKAN. Specifically, I have developed and published the extension `ckanext-dataontosearch` on GitHub² and PyPI³.

Organization

DataOntoSearch itself is not put into CKAN. Instead, API endpoints have been added to DataOntoSearch, which are then used by the CKAN extension. DataOntoSearch and CKAN may therefore run on different servers, and one instance of DataOntoSearch can serve multiple CKAN instances. This provides flexibility, and also makes it easier to add DataOntoSearch to existing CKAN instances, which may not have the technical know-how necessary to set up DataOntoSearch.

The CKAN extension is split into two plugins, which can be activated individually. One of them provides the ability to manually associate datasets with concepts. The other plugin adds the ability to perform semantic search inside CKAN. The split helps with organizing the code, and also makes it possible for a CKAN instance to e.g. just use the semantic search, and not the dataset tagger.

CKAN is organized into actions, routes and templates, which more or less correspond to the model, controller and view of the Model-View-Controller architectural pattern. This structure is adhered to by the extension as well, which adds new actions, routes and templates to CKAN.

- Each **action** is made easily available to the routes, and is also automatically made available through CKAN's own APIs. Each action also has a separate authorization function associated with it, which decides if the user is allowed to perform the action.
- Each **route** is mapped to a set of URLs, and is invoked when the user visits one of the matching URLs. The CKAN project is currently switching to Flask (a Python

¹<https://github.com/tobinus/OTD-semantic-framework>

²<https://github.com/tobinus/ckanext-dataontosearch>

³<https://pypi.org/project/ckanext-dataontosearch/>

library handling HTTP requests and mapping them to Python functions) for handling routes, and the extension therefore uses the Flask capabilities exposed by CKAN. The routes are responsible for collecting data and invoking the different actions, before letting the template generate the HTML.

- Each **template** is used by one or multiple routes, and make up the user interface of CKAN. A flexible organization is adapted, which allows extensions to override small “snippets” and therefore add a link here and another link there, so the extension can change how the existing pages in CKAN look.

The different ways the CKAN extension alters CKAN are described below, following this distinction between actions, routes and templates.

Actions

The DataOntoSearch plugin adds the following actions:

- Concept list:** Fetches a list of concepts and their labels from DataOntoSearch.
- List all tags:** Fetches a list of dataset-concept associations, for all datasets.
- List tags:** Fetches a list of dataset-concept associations, for the specified dataset.
- Create tag:** Add a dataset-concept association, adding the dataset to DataOntoSearch’s index if it’s not there already.
- Delete tag:** Remove a dataset-concept association.
- Delete dataset:** Remove all dataset-concept associations for the specified dataset, and remove the dataset from DataOntoSearch’s index.
- Search datasets:** Perform a semantic search using DataOntoSearch.

These actions are invoked by the routes when required. For example, the action for listing concepts is used when the user loads the form for editing concepts, since a list of concepts available is presented there. Additionally, the “Delete dataset” action is set to automatically run when a dataset is removed from CKAN, to avoid DataOntoSearch presenting datasets that no longer exist.

Protection is added so that only users who would normally be able to edit a dataset, can edit the dataset’s associated concepts. There is also a weak protection added so that the semantic search only shows datasets the user is authorized to see. This protection is weak because using the API or GUI of DataOntoSearch directly, you can see all datasets, irrespective of their privacy settings in CKAN.

Routes

New routes are also added, providing the DataOntoSearch functionality inside CKAN:

/dataontosearch/tagger/<dataset_id>/ Shows the concepts associated with the dataset.
This view is shown in Figure 6.1.

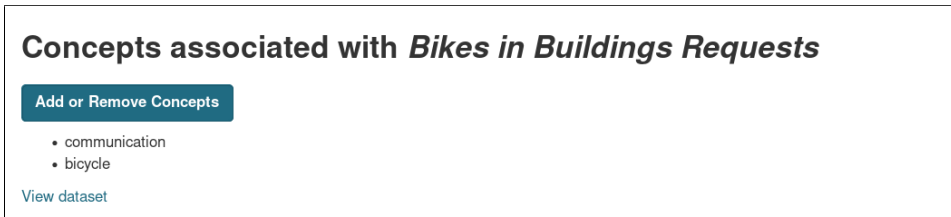


Figure 6.1: A list of concepts are shown for this dataset.

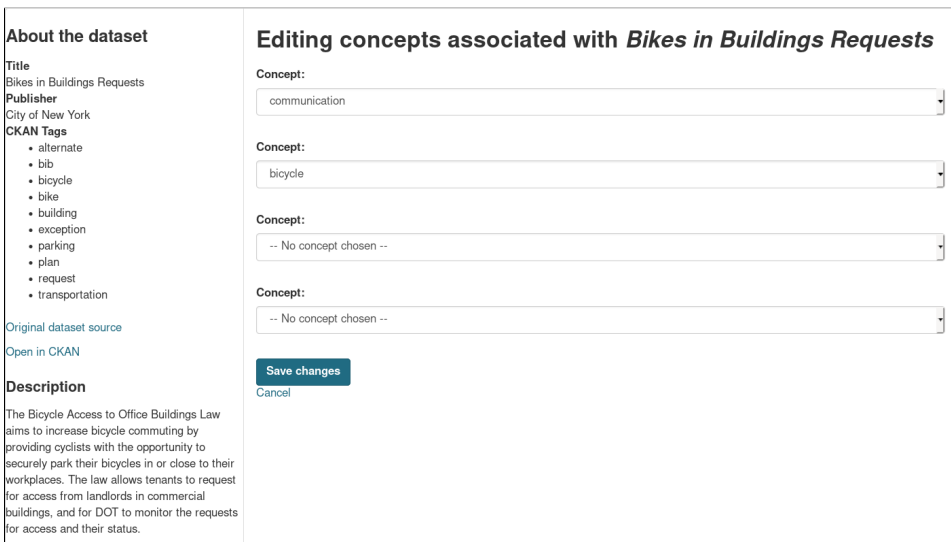


Figure 6.2: The user can add, change or remove concepts for the selected dataset.

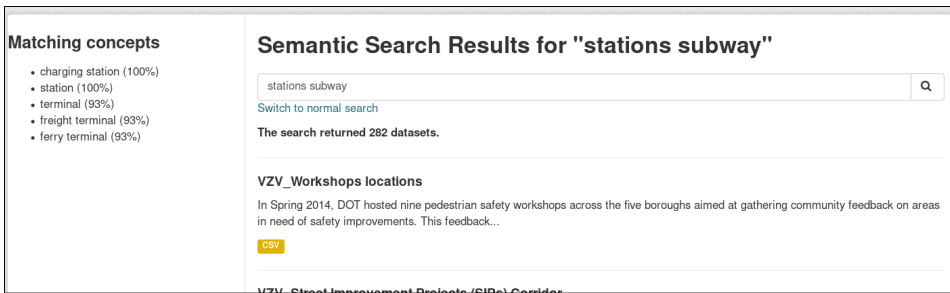


Figure 6.3: The semantic search page lists matching concepts on the left, and the search results in the main part of the page.

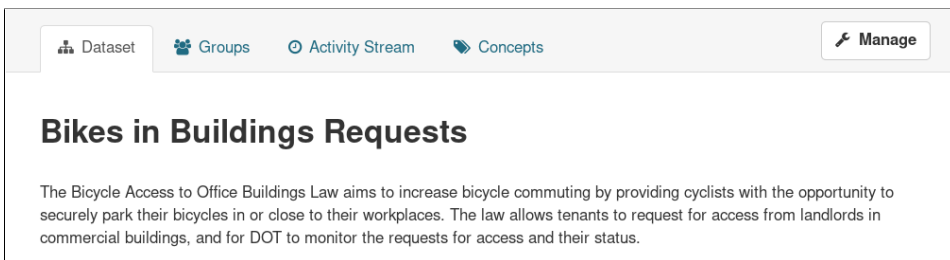


Figure 6.4: A link to view associated concepts is shown when viewing a dataset in CKAN.

`/dataontosearch/tagger/{dataset_id}/edit` Lets you add, remove and change what concepts are associated with the dataset. Figure 6.2 shows an example.

`/dataontosearch/search` Provides semantic search results, as seen in Figure 6.3.

Templates

Each route has a template associated with them, which implements the route’s graphical user interface. In addition, some small changes are made to existing “snippets” in CKAN, in order to make the new routes discoverable by users:

- When you view a dataset, a new tab links to the route listing the dataset’s concepts. This is shown in Figure 6.4.
- When you edit a dataset, a new tab links to the route where you can edit the dataset’s concepts, as shown in Figure 6.5.
- When you make a search using the built-in CKAN search, a link lets you repeat that search using the semantic search. Figure 6.6 showcases this.

Installation

The recommendations for CKAN extensions are followed. Specifically, the extension is published onto PyPI, which lets CKAN administrators install the extension by simply using the Python packaging tool `pip`. Afterwards, the administrator must follow the same

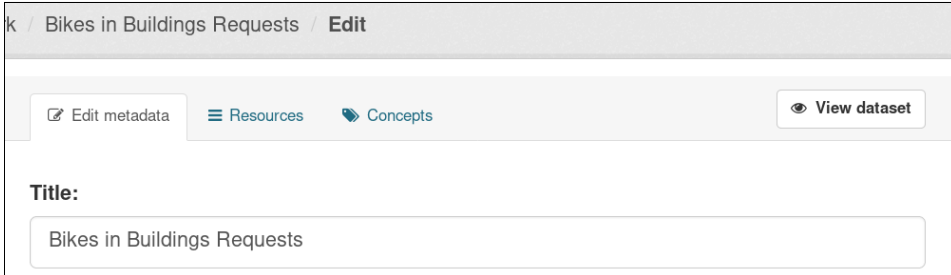


Figure 6.5: A link to edit associated concepts is shown when editing a dataset in CKAN.

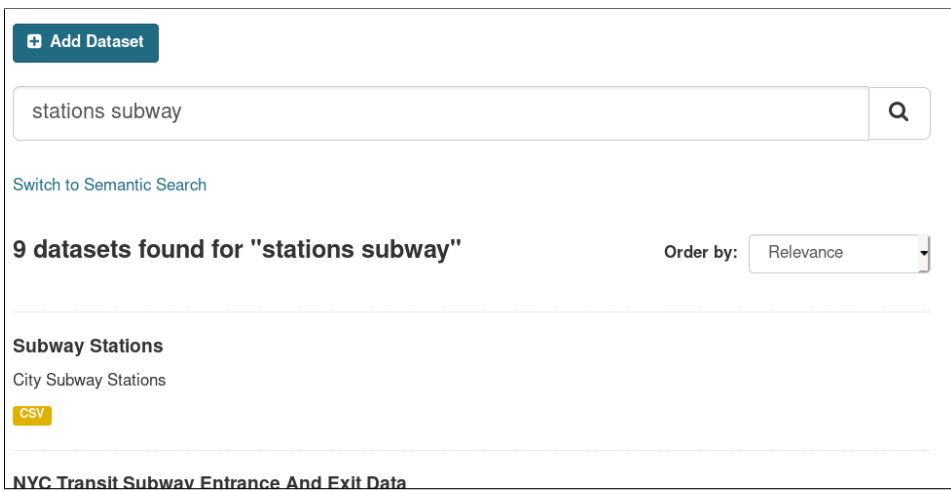


Figure 6.6: When using the CKAN search, the user can click a link to perform the same search using DataOntoSearch's semantic search.

procedure as for every other extension, meaning it must be added to the configuration file. The DataOntoSearch extension requires some extra configuring, e.g. so the extension knows where to find the DataOntoSearch instance. The specific instructions are presented in the extension's README file, which is included in Appendix C.4, and are based on the template CKAN provides to plugin authors.

6.2.2 API

When a separate system, like the CKAN extension just discussed, wishes to interact with DataOntoSearch, then that is usually done through APIs. They provide a well-defined, predictable way of interacting with the application.

DataOntoSearch version 1 did not have any functioning APIs. Its dataset tagger did have some rudimentary APIs defined, but they had not been updated to reflect a new storage structure that had been adapted. The semantic search did not have any APIs exposed on the web. I have therefore dedicated a good amount of work towards implementing APIs for DataOntoSearch, mostly informed by the needs of the CKAN extension.

Handling multiple instances

By keeping CKAN and DataOntoSearch separate, we are able to serve multiple CKAN instances with just one instance of DataOntoSearch. This, however, presents a challenge: How do you avoid different instances interfering with each other? E.g., how do you avoid that when a user performs a semantic search on instance A, they get datasets from instance B?

From before of, DataOntoSearch uses a number of different graphs. They are:

1. **Ontology:** Concepts and their relation to one another.
2. **Dataset:** Available datasets in DCAT format (the “index,” if you like).
3. **Similarity:** Similarity graph, storing dataset-concept associations made manually.
4. **Autotag:** Similarity graph, storing dataset-concept associations made by the auto-tagger.

When you search, you need to have selected one graph of each graph type. This would quickly get unwieldy for API users for two reasons:

1. Needing to send four graph IDs with each API call adds complexity.
2. Not all combinations of graphs work. Specifically, the similarity and autotag graphs depend on one particular ontology and one particular dataset graph, since they define associations between entities in the two graphs.

Therefore, a new entity has been introduced, called *Configuration*. Each Configuration simply holds an ID of four graphs that are used together, along with a label and the date it was last modified. API users simply provide the ID of the Configuration they want to use. Based on what Configuration they specify, DataOntoSearch knows exactly which graphs to use.

In practice, each CKAN instance should get one Configuration each. Though the Ontology graph will likely be shared between instances, the other graphs should be unique. This way, the instances' indexed datasets and stored dataset-concept associations are safe from others.

Design

The principles of REST are used to a great degree in designing the HTTP APIs. They are widely used on the web, and are intended to make it easier to use such APIs. Some of the principles of REST are [18]:

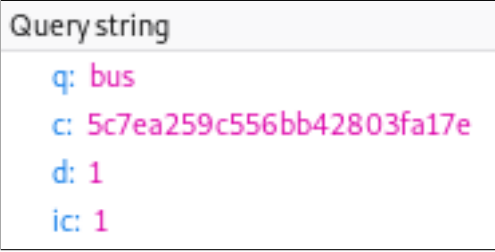
- Stateless requests and responses, so they are self-contained and do not depend on earlier interactions.
- The data is represented in a standard format, which need not be the same as the internal representation.
- The inner workings of the application is encapsulated, so users of the API need only concern themselves with the API's constraints and semantics.
- Different URLs reflect different *resources*, just like different URLs on the web refer to different articles and so on.
- When applied to HTTP, the so-called HTTP method is used to indicate the type of action you'd like to perform, as specified by the protocol. GET means retrieving a resource without causing side-effects, POST is typically assigned the meaning of creating new resources, DELETE is used to delete a resource, and so on [17].

DataOntoSearch's functionality is split across two webservers, which is a decision that was made already for DataOntoSearch version 1. The dataset tagger is responsible for viewing and editing the dataset-concept associations. The search server is responsible for answering search queries. The APIs are therefore also split across the two webservers, so the code can be re-used and so the structure is consistent.

For the dataset tagger, multiple endpoints are provided, which more or less correspond one-to-one with the actions described for the CKAN extension. They are documented in the code repository. The documentation is also included as Appendix C.2. The endpoints are:

Method	Endpoint	Purpose
GET	/api/v1/<configuration>/concept	Retrieve concepts available to you
GET	/api/v1/<configuration>/tag	Retrieve existing associations between dataset and concepts
POST	/api/v1/<configuration>/tag	Tag a dataset with a concept
DELETE	/api/v1/<configuration>/tag	Remove a tag connecting a concept to a dataset
DELETE	/api/v1/<configuration>/dataset	Remove a dataset and all associated tags

For those endpoints, the Configuration chosen is specified in the URL. This means that it is possible to create access control rules for individual Configurations in the webserver



```
Query string
q: bus
c: 5c7ea259c556bb42803fa17e
d: 1
ic: 1
```

Figure 6.7: Query parameters used to make an example request to the `/api/v1/search` endpoint. `q` is the query, `c` is the Configuration chosen and `d` and `ic` affect what information is included in the response. The documentation in Appendix C.3 describes all available parameters.

software (e.g. Nginx or Apache). For example, you can specify rules that only allow one particular login for paths starting with `/api/v1/5c7ea259c556bb42803fa17e`, while another login is used for paths starting with `/api/v1/5ccb0bdc556bb13f71377d7`, corresponding to two different Configuration instances. The `v1` in the endpoint path refers to the version of the API, not the version of DataOntoSearch.

There is only one endpoint for the search webserver, namely a GET endpoint for `/api/v1/search`. The query and a number of optional options, such as the Configuration to use, are set using query parameters. An example of an API request is shown in Figure 6.7, along with the response in Figure 6.8 on page 81. The endpoint is documented in the code repository, though the documentation is included in Appendix C.3.

6.2.3 Improving how the query is associated with concepts

When I experimented with DataOntoSearch version 1, I quickly noticed that the concepts that my query matched with didn't make much sense. Even after I had implemented search suggestions based on concept labels, selecting a suggestion would not guarantee that the underlying concept was associated with the query.

Even more concerning was the fact that many concepts seemed inaccessible. I have implemented a route that lets you see the similarity of your query to all the concepts, accessible at `/scores`. Experimenting with this tool, I could see that many concepts had a score of 0.000, which almost never happens when a synset is found for both the concept label and the query. This indicated that there were many concepts that did not match with any WordNet synsets. This was a serious problem, since synsets are used as a common link between queries and concepts.

In short, the way I saw it, there were two desired properties of DataOntoSearch that it did not possess:

- If you use a concept's label as your search query, you should match with that concept.
- For all concepts, there should exist a query that matches with that concept.

To solve the first problem, the code that compared the query to concepts was modified to use the same code for processing the natural language of the query and the concepts.

This way, if it receives the same input – i.e. the query is equal to a concept’s label – it will find the same WordNet synsets for them both, and they will compare equal. This did, however, uncover some more problems with system.

The code implementation concerned with the issues discussed here can be found in Appendix D, specifically in Appendix D.2 and D.3.

Increasing specificity of matching concepts

With concept labels being split up, it became difficult to match with one-word concepts whose word was also a part of longer concept labels.

Take *bus*, for instance. Now, if you searched for *bus*, you got a perfect match with:

- bus
- bus stop
- bus terminal

This led to a lot of noise in the results, since you could not match with the *bus* concept without simultaneously matching with the other concepts.

The reason this was the case, was that both the query and the concept labels were matched with WordNet synsets that were put into one big bag, one for the query and one for the concept. All pairs of synsets were compared to examine their semantic similarity, using the graph structure of WordNet. The similarity between the most similar synsets was taken to be the similarity score between the query and that concept.

Though this approach makes sense for single-word concept labels, it breaks down when concepts can have labels with multiple words. Since all synsets from all words are put into a single set of synsets, both *bus stop* and *bus terminal* will include the same synsets as *bus*, since the word “bus” is present in all three, and therefore produces the same WordNet synsets.

This situation is comparable to using term frequency (TF) as an information retrieval technique without normalizing it. In short, not applying normalization would mean making long documents more likely to be matched, just because they include more words and therefore have a greater chance of mentioning a word from the user’s query. By dividing the raw term frequency by the number of terms in the document, smaller but relevant documents actually stand a chance against longer documents that just happen to mention the search term in passing.

The solution I implemented here, was to modify how synsets were returned from the algorithm that matches concept labels with WordNet synsets. Instead of returning just one set of synsets, it now returns one set of synsets *per word* in the label. For each set of synsets, the comparison is still the same: The most similar pair of synsets from the entire query and the particular word in the label is chosen. Thus, a similarity score is calculated for each word in the concept label.

The previous behaviour can, at this point, be simulated by picking the highest similarity score found among the words. The new behaviour, however, uses the *harmonic mean* across all the words. Thus, when you search for *bike*, the similarity to the concept *bicycle* is set to 1, but the similarity to *bike sharing* is just 0.688.

Of course, the same problem is present when the *query* contains words that map with both long and short concept labels. Searching for *bike sharing* therefore gives a perfect match with both *bike sharing* and *bicycle*. I experimented with penalizing concepts that matched with only a portion of the query, but the few experiments gave poor results.

Making all concepts available

When the user's query is compared to concepts, a number of different NLP methods are used. However, in the end, WordNet is used to compare the semantics of the words.

The way this process was set up in DataOntoSearch version 1 meant that the query was split into tokens that were put into WordNet one by one. For the concepts, however, their entire label was put right into WordNet, without splitting it up. It turns out that even though WordNet has entries that consist of multiple words, you must use underscores between the words when looking them up. Since spaces were used in DataOntoSearch version 1, any concept that only had labels made up of multiple words never matched with any synonym sets in WordNet and were consequently inaccessible and not possible to match with.

This is fixed in version 2, at first by processing concepts the same way the query is processed.

Increasing the search's precision with word-bagging

There is an element of inaccuracy introduced when splitting words, though: Even if the user searched with a phrase known to WordNet, it was split up into the individual words and used one-by-one in WordNet. Thus, phrases which could be assigned one specific semantic meaning in WordNet, were instead assigned a high number of synonym sets.

For example, querying WordNet for synsets using the phrase *bus stop* yields just one synset, with the definition "a place on a bus route where buses stop to discharge and take on passengers." However, *bus* and *stop* separately are mapped to 7 and 22 synsets respectively. DataOntoSearch does not know how to disambiguate between them and will try using all the 29 synsets, using whatever synset gives the highest similarity to a concept.

Clearly, it would be best if multi-word phrases in the query and concept labels could be matched with multi-word phrases in WordNet. Before implementing this, an analysis was done of the WordNet synsets available, to figure out exactly how large multi-word phrases should be tried out. When only supporting single-word phrases, 67% of the WordNet synsets can be matched with. Adding support for bigrams and trigrams (bags of two and three words, respectively) increases this number to 99%. Supporting longer phrases does not give big returns, so the upper bound is set to supporting trigrams.

A greedy matching algorithm is implemented. It first tries all trigrams. If a trigram is found in WordNet, the tokens that made up that trigram are replaced by dummy tokens in the list of tokens. Next, all bigrams are tried out, skipping any bigrams that include a dummy token. The tokens that made up any matching bigrams are also replaced by dummy tokens. Finally, WordNet is queried using the remaining single words. The final list of synset groups is returned.

A problem with this approach is that it clashes with the NLP methods used. For example, the stop-word remover removes words that may be integral to a phrase, like *of* in *time of day*. The remaining phrase, *time day*, does not make sense and does not match with

anything in WordNet. The stop word remover was therefore removed. Most stop words are not represented in WordNet anyway, since e.g. prepositions are left out.

Another problem is that only certain parts of speech are preserved while others are eliminated, which also has the potential to cut out phrases known to WordNet. The lemmatizer used is also unnecessary, since the WordNet library carries out lemmatizing automatically when looking up entries. The lemmatizer was therefore cut out, and the constraints placed on what phrases were included were relaxed.

Decreasing the number of synsets to compare

As was mentioned earlier, a concept's similarity to the query is found by comparing all WordNet synsets generated from the concept labels to all the synsets generated using the query. As the number of synsets gets higher, the process time for this comparison process increases rapidly, since the time complexity is bound by $n * m$, where n and m are the number of synsets generated from the query and concepts. Reducing the number of synsets in the query by half therefore cuts the processing time by a factor of four.

In DataOntoSearch version 1, a part of speech tagger is used to figure out what part of speech each token is. It uses information about the token's surrounding words to determine what part of speech it is. This information was only used to filter out words that were not deemed interesting, before the part of speech tag was discarded.

In version 2, the part of speech tag is kept. It turns out that the WordNet library supports specifying a part of speech when you try to find synsets. It acts as a filter, ensuring that only synsets of the corresponding part of speech are retrieved. This greatly reduces the number of synsets any particular word is matched with, and also increases the accuracy.

The part of speech tagger is not able to disambiguate between different possible parts of speech when there is only one word. It is therefore skipped for those cases, so that all possible WordNet synsets are used. This was necessary because it would arrive at the wrong conclusion some of the time.

6.2.4 Increasing the ontology's importance

When the query is compared with datasets, its similarity with concepts is used. This is called the Query-Concept Similarity Vector (QCSV). Similarly, the datasets' similarity to concepts is stored in the Dataset-Concept Similarity Matrix (DCSM).

Cosine similarity is used to compare the QCSV with each vector of the DCSM, with each concept being one dimension. Concepts that are associated with both the query and the dataset increase the similarity score output by the cosine similarity, while differences are penalized.

In DataOntoSearch version 1, the ontology is used to create the Concept-Concept Similarity Matrix (CCSM). The matrix is used to create the DCSM, by enriching the existing associations (the automatic or manual tagging) with new dataset-concept relations, based on the new concepts' similarity to the concepts already associated with the dataset. The structure of the ontology therefore affects all the similarity scores and their distribution.

On the other hand, the CCSM is never used with the query. Its similarity to concepts is only defined through the similarity scores calculated by WordNet, using the same algorithm used for creating the CCSM, but with the graph structure of WordNet synsets rather

than the ontology. This graph consists of WordNet synsets as nodes. Relationships between the synsets, like one synset being more general than another, make up the edges of the graph. In many ways, WordNet is used as if it was an ontology, though this is not the purpose it was made for.

Due to the way cosine similarity works, and the way the similarities are calculated, this has a rather interesting consequence: *A dataset is more similar to the query if the structure of its concepts in the ontology is more similar to the structure of the synsets in WordNet.*

To avoid problems with datasets' ranking being determined by this irrelevant factor, a new step is introduced.

1. The query is associated with concepts through WordNet, just like before.
2. A new threshold called T_Q is used. Only similarities that are $\geq T_Q$ are kept.
3. The Query-Concept Similarity Vector is enriched using CCSM in the exact same way the datasets were enriched when creating the DCSM.
4. Just as for the datasets, the T_C threshold is now used here to ensure low-scoring concepts do not affect the cosine similarity. The same threshold is used for the query and datasets to ensure consistency between them.

The effect of this change is that WordNet's involvement is reduced to simply finding the *most* similar concepts. Once they are found, the ontology takes over. This way, the structure of WordNet no longer affects what datasets are considered most similar. The code implementation of this can be seen in Appendix D.1.

6.2.5 New threshold variables

Hagelien's thesis [20] and the paper describing DataOntoSearch [25] both describe two threshold values that can be varied to tune DataOntoSearch's behaviour. However, the descriptions did not match up with the code that was made available on GitHub, where only one of the thresholds were implemented, and it was not implemented in a way that let you vary it.

When I started working on the code needed to evaluate DataOntoSearch version 1 for the pre-study (RQ1), I decided to not spend time implementing the two thresholds, since I did not understand them at the time. Later, when I worked on evaluating version 2, I had to search up how the variables work.

In addition to the aforementioned thesis and paper, I asked Hagelien directly by email about the distinction between the two thresholds. His reply came with a warning that he might be misremembering things, but I include it here because it is interesting. Table 6.3 on the next page shows what the different sources had to say about the thresholds.

Hagelien's thesis has a description of T_C which corresponds with the paper, since it relates the threshold to "concept relevance." Concept relevance is described as "a limit for how similar a concept and a dataset must be for a concept to be considered relevant." However, the thesis also describes T_C as a threshold similar to T_S , stating that "[i]f a dataset has a similarity score higher than the threshold, the dataset is expected to [be a] part of the search result." It goes on to describe how T_C determines whether a dataset

Table 6.3: Previous descriptions of T_C and T_S

Source	T_C	T_S
Thesis [20]	Contradicting descriptions	Lower threshold for dataset-query similarity
Paper [25]	Concepts w/ sim. $\geq T_C$ are associated with the dataset w/ score = 1	Lower threshold for dataset-query similarity
Email	Concepts w/ sim. $\geq T_C$ are associated with the dataset	Concepts w/ sim. $\geq T_S$ are associated with the query

is “correct” or not, depending on its similarity to a concept. I find these descriptions contradictory.

With DataOntoSearch version 2, I decided to implement the following thresholds:

T_C Concepts with similarity $\geq T_C$ are associated with the dataset (as described in Hagelien’s email).

By keeping their similarity score and not setting it to 1, we can calculate the cosine similarity with greater granularity.

T_S Lower threshold for dataset-query similarity (as described in the thesis and the paper).

T_Q New threshold. Concepts with similarity $\geq T_Q$ are associated with the query.

I was inspired to implement this when I saw it described as T_S by Hagelien in his email, though I made it a new threshold. This is implemented as a part of the previously described effort in increasing the ontology’s importance.

The threshold T_C is used when generating the Dataset-Concept Similarity Matrix. Since this is an offline process, there exists no good way of specifying T_C when using the online search. It is supported when using the command-line tools, though, including the evaluation tools.

Both T_S and T_Q can be varied on the fly. Though the graphical user interface does not provide any way of changing them, users of the API may change them as they wish. They are, of course, supported in the command-line interface.

The evaluation includes an analysis of how the different thresholds affect the search results.

6.2.6 Autotagging

When Hagelien implemented the autotagging functionality, he helped make it possible to perform ontology-based semantic search on datasets without the need for manually associating datasets with concepts. Though this is very helpful, I have not focused on improving the autotagger in my work. The main reason is that I have focused on making it easier to perform aforementioned manual linking, which should hopefully alleviate some of the need for the autotagger.

The one change I have made, is making it possible to choose between the English WordNet and the Norwegian OrdVev. This makes it possible to use the autotagger with

datasets that have their metadata written in English. Since the datasets used by Hagelien were from the Open Transport Data project, they were written in Norwegian and there was therefore no need to support English metadata.

6.2.7 Manual tagging process

The manual tagging process used to be a very labour intensive process. Though spreadsheets were used to write down what concepts each dataset should be associated with, there was no easy way to generate the spreadsheets, and there was no automatic way of parsing them and create the RDF graph structure needed to represent the associations.

Through my work, I have added the ability to generate a spreadsheet fit for manually linking many datasets at once. Each row of the spreadsheet represents one dataset, and contains the dataset's title, description and ID. It can be used in one of two ways:

- **One column per concept:** All the concepts have one column dedicated to them. If you write anything in the table cell corresponding to the row of dataset *d* and the column of concept *c*, an association will be made between *d* and *c*. An example of how this looks is shown in Figure 6.9
- **One column with concepts:** You can create a column with the header “Concepts.” In it, you write the name of any concepts that the dataset on that row should be associated with. Multiple concepts are separated with comma and white-space.

A new utility has been made for parsing a CSV and turning it into an RDF graph of manual associations. There will likely be errors in the CSV file, such as misspellings or use of alternate names of concepts. Using the `--check` flag, all unrecognized concepts are reported at once, along with the names of the cells where they appear.

All in all, these utilities enables mass-tagging of datasets and concepts, while reducing the effort involved in setting up and getting the results into the system.

The previously described CKAN extension adds another way of manually linking datasets and concepts. When using it, any unrecognized datasets are also added to DataOntoSearch. In other words, if you start with an empty CKAN and empty dataset and similarity graphs in DataOntoSearch, the two systems will stay in sync with each other.

This approach works best for incremental changes, e.g. adding concepts whenever you upload a new dataset. When tagging multiple datasets in one go, the slow response times and the many clicks needed to navigate around make it less enjoyable.

There was a rudimentary web interface for adding manual links in DataOntoSearch version 1. As a part of the API implementation, I also updated this web interface so it actually works, and is compatible with Configurations. That said, it provides very little feedback about which associations already exist, and it requires that you write the URL of the dataset. The CKAN extension, on the other hand, gives a clear view over what concepts the dataset is associated with, and you only need to find the dataset inside CKAN.

6.2.8 Web interface

The web interface was updated slightly in preparation for the usability tests in RQ1, as discussed in Section 5.2.4. I have not focused on improving it further after that, because I

wanted to focus on integrating the search into CKAN instead.

This approach essentially means that a new user interface has been created, namely the extension's semantic search web page in CKAN. Since it follows the look and feel of CKAN itself, it is more stylistic and should hopefully leave a better impression than the built-in web interface of DataOntoSearch.

That said, it does not show the concepts that are associated with each dataset yet. It also does not suggest concept labels when typing the query. In those areas, the built-in web interface of DataOntoSearch is still better.

```

JSON
▼ concepts: [...]
  ▼ 0: {...}
    label: bus
    similarity: 1
    uri: http://www.qaat.com/ontologies#Bus
  ▼ 1: {...}
    label: car
    similarity: 0.96
    uri: http://www.qaat.com/ontologies#Car
  ▶ 2: {...}
  ▶ 3: {...}
  ▶ 4: {...}
▼ results: [...]
  ▼ 0: {...}
    ▼ concepts: [...]
      ▼ 0: {...}
        label: information
        similarity: 1
        uri: http://www.qaat.com/ontologies#Information
      ▶ 1: {...}
      ▶ 2: {...}
      ▶ 3: {...}
      ▶ 4: {...}
    description: This list contains historical information on the status of current medallion vehicles authorized to operate in New York City. This list is accurate to the date and time represented in the Last Date Updated and Last Time Updated fields. For inquiries about the contents of this dataset, please email licensinginquiries@tlc.nyc.gov. To view the latest list please visit https://data.cityofnewyork.us/Transportation/Medallion-Vehicles-Authorized/rhe8-mgbb/data.
    score: 0.9711075519890722
    title: Historical Medallion Vehicles - Authorized
    uri: http://192.168.56.101:8000/dataset/d56cf0be-f09f-47ae-8112-2386593fce74
  ▼ 1: {...}
    ▶ concepts: [...]
    description: TLC authorized For-Hire vehicles that are active. This list is accurate to the date and time represented in the Last Date Updated and Last Time Updated fields. For inquiries about the contents of this dataset, please email licensinginquiries@tlc.nyc.gov.
    score: 0.9707770119711832
    title: For Hire Vehicles (FHV) - Active
    uri: http://192.168.56.101:8000/dataset/48f5c012-b5fa-4f51-b916-2b32a92176e1
  ▶ 2: {...}

```

Figure 6.8: A small sample of the response to the API request shown in Figure 6.7. The documentation in Appendix C.3 describes the meaning of all the fields.

Chapter 6. Implementation and results of RQ2

1	A	B	C	B1	B4	B41	B5
1	Dataset	Title	Description	Regulation	License	Drivers	Licenses*Permit
2	http://192.16	Driver License, Permit, and Non-Driver Identification Cards issued by County, Age, and Gender: 2011 - 2015	Annual data on the number of DMV issued photo document holders broken out by County of residence, age, and gender.		X		
3			The For-Hire Vehicle (FHV) trip records include fields capturing the dispatching base license number and the pick-up date, time, and taxi zone location ID (shape file below). These records are generated from the FHV Trip Record submissions made by bases. Note: The TLC publishes base trip record data as submitted by the bases, and we cannot guarantee or confirm their accuracy or completeness. Therefore, this may not represent the total amount of trips dispatched by all TLC-licensed bases. The TLC performs routine reviews of the records and takes enforcement actions when necessary to ensure, to the extent possible, complete and accurate information.				
4	http://192.16	2016 For Hire Vehicle Trip Data	For trip record data including TLC taxi zone location IDs, location names and corresponding boroughs for each ID can be found here. A shapefile containing the boundaries for the taxi zones can be found here.				
5	http://192.16	Traffic Control Device Inventory	The New York State Department of Transportation (NYSDOT) traffic control devices data set is a list of all of the traffic devices that are either owned or maintained by NYSDOT. The devices included are of various types such as traffic signals, street lights, beacons, flashers, navigational lights, Intelligent Transportation Systems (ITS), etc. Devices in the 5 boroughs of New York City are not owned or maintained by NYSDOT and therefore not represented in this dataset				
6	http://192.16	Commissioner's Correspondence	DOT receives, tracks, and responds to correspondences sent to the Commissioner's Correspondence Unit (CCU) and Borough Commissioner's (BC) offices. This includes some requests initially sent to 311 and the Mayor's Office, and then forwarded to DOT.	X			
7	http://192.16	Traffic Volume Counts (2012-2013)	Traffic volume counts collected by DOT for New York Metropolitan Transportation Council (NYMTC) to validate the New York Best Practice Model (NYBPM).				
8	http://192.16	For Hire Vehicles (FHV) - Active	TLC authorized For-Hire vehicles that are active. This list is accurate to the date and time represented in the Last Date Updated and Last Time Updated fields. For inquiries about the contents of this dataset, please email licensinginquiries@tlc.nyc.gov .		X		
9			The yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery				

Figure 6.9: Example of how the generated CSV file can be used for manual tagging.

6.3 Results

The results from the evaluation done after the improvements were implemented are presented in this section. They answer the question of how DataOntoSearch reacts to different threshold values, and how the new version fares compared to both the old and Google Dataset Search.

In the following, the tables are coloured. The intensity of the colour indicates how high the number in that cell is. The colour itself indicates what measure is being reported.

6.3.1 Sensitivity of threshold variables

As explained in Section 6.2.5, there are three thresholds which can be varied in DataOntoSearch version 2. They are:

T_S : Only datasets with a query similarity score $\geq T_S$ are retrieved.

T_Q : Only concepts whose similarity to the *query* is $\geq T_Q$ are associated with the query, before applying the CCSM.

T_C : Only concepts whose similarity to a *dataset* or the *query* is $\geq T_C$ are associated with that dataset or query after having applied the CCSM.

Effect of varying T_S The way T_S affects DataOntoSearch’s performance can be seen in Table 6.4 on page 85. The measures are averaged over all combinations of $T_C, T_Q \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. Since this threshold is a cut-off point for at which point no more datasets will be retrieved, it naturally affects recall the most. When set to 0.00, all datasets are retrieved for every query and we therefore get a recall of 100%, and when set to 1.00, almost no dataset is retrieved, so recall is close to 0%. How similar the datasets are to the query’s concept vector will generally differ depending on the values of T_C and T_Q , so looking at T_S averaged over all values of T_C and T_Q may not give us much information.

Effect of varying T_Q Moving on to T_Q , its effect on DataOntoSearch’s performance can be seen in Table 6.5 on page 85. The measures are averaged over all combinations of $T_S, T_C \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. There does not seem to be any pattern in what values give the best performance, since a higher value is good for the manually tagged dataset-concept associations, while a score closer to the middle is marginally better for the automatic tags.

Effect of varying T_C Finally, the effect of varying T_C is shown in Table 6.6 on page 86. The measures are averaged over all combinations of $T_S, T_Q \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. The trend seems to be better performance for lower values of T_C , though with a surprising performance increase for 1.00 when using the manual tagging.

Effect of varying T_Q and T_C together We now move on to looking at multiple variables in tandem. First, the F1 scores and MAP values for combinations of T_Q and T_C are shown in Table 6.7 on page 86 and Table 6.8 on page 91 respectively. The measures are averaged over all $T_S \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. Recall that the F1 score is a combination of precision and recall. There does not seem to be any apparent pattern in what threshold values give the best F1 score for the automatic tagging, which may indicate that T_S is a greater contributor to the F1 score than T_Q and T_C . For the manual tagging, the score increases when both are close to 1.00.

Continuing to MAP, which is a measure of how well the datasets were ranked, there seems to be a pattern of lower scores of both increasing the MAP measure slightly. However, setting T_Q to 1.00 yields a high boost, but only for the manual tagging. For the automatic tagging, setting $T_Q = 1.00$ negatively affects performance. With $T_Q = 1.00$, the system seems to perform better when T_C is not in the middle.

Effect of varying T_S over combinations of T_Q and T_C We finally consider all the three variables at once. To make this possible, the results have been split up over multiple tables. We once again use the F1 score and MAP measures. The F1 score is presented for automatic tagging in Table 6.9 on page 92 and for manual tagging in Table 6.10 on page 93. There is a tendency towards higher values when T_S is set to 0.25, though this is not so apparent for the automatic tagging. There are some specific combinations that yield much greater results than neighbouring combinations, namely $T_S = 0.25, T_Q = 0.75, T_C = 1.00$ and $T_S = 0.25, T_Q = 1.00, T_C = 1.00$. This is true for both the automatic and the manual tagging, though the effect is much greater for the manually derived associations.

The MAP measurements are shown for automatic tagging in Table 6.11 on page 94 and for manual tagging in Table 6.12 on page 95. Here, $T_S = 0.00$ seems to be the best performing assignment, though not much is lost when setting $T_S = 0.25$. The best performing combinations for the F1 score perform reasonably well for MAP as well, especially $T_S = 0.25, T_Q = 1.00, T_C = 1.00$ for the manual tagging.

6.3.2 Search engine comparison

One of the main goals of this evaluation is to demonstrate how the new version of DataOntoSearch compares to Google Dataset Search and DataOntoSearch version 1. We will first arrive at a set of threshold values to use for DataOntoSearch version 2, before presenting the evaluation results.

Before that, though, it must be noted that Google Dataset Search seemed to have a bug the day the queries were run (May 31st), which meant that e.g. the dataset with subway station data was unavailable when using the “site:data.ny.gov” operator, despite the dataset showing “data.ny.gov” as its source. This was confirmed with manual use of the search afterwards the same day. A couple days later (June 3rd), the bug seems to have been fixed. Google would likely perform much better if rerun today, though it has not been possible to do this due to time constraints.

Table 6.4: Effect of varying T_S . The measures are averaged over the other two thresholds, using all combinations of values $\in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

Tagging	T_S	Precision	Recall	F1 score	R-Precision	MAP
auto	0.00	6 %	100 %	9 %	11 %	13 %
	0.25	7 %	79 %	9 %	10 %	11 %
	0.50	5 %	67 %	7 %	8 %	9 %
	0.75	4 %	54 %	6 %	6 %	7 %
	1.00	0 %	0 %	0 %	0 %	0 %
manual	0.00	6 %	100 %	10 %	17 %	22 %
	0.25	10 %	77 %	13 %	14 %	17 %
	0.50	9 %	59 %	11 %	11 %	14 %
	0.75	6 %	40 %	7 %	8 %	9 %
	1.00	2 %	1 %	1 %	1 %	1 %
Average		5 %	58 %	7 %	8 %	10 %

Table 6.5: Effect of varying T_Q . The measures are averaged over the other two thresholds, using all combinations of values $\in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

Tagging	T_Q	Precision	Recall	F1 score	R-Precision	MAP
auto	0.00	4 %	67 %	7 %	7 %	8 %
	0.25	4 %	66 %	7 %	7 %	8 %
	0.50	4 %	64 %	7 %	7 %	9 %
	0.75	5 %	59 %	7 %	9 %	9 %
	1.00	4 %	43 %	5 %	5 %	6 %
manual	0.00	4 %	55 %	6 %	5 %	7 %
	0.25	5 %	55 %	6 %	5 %	7 %
	0.50	4 %	55 %	6 %	5 %	8 %
	0.75	9 %	59 %	11 %	12 %	15 %
	1.00	11 %	53 %	12 %	22 %	25 %
Average		5 %	58 %	7 %	8 %	10 %

Table 6.6: Effect of varying T_C . The measures are averaged over the other two thresholds, using all combinations of values $\in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

Tagging	T_C	Precision	Recall	F1 score	R-Precision	MAP
auto	0.00	4 %	76 %	7 %	8 %	10 %
	0.25	4 %	75 %	7 %	8 %	10 %
	0.50	4 %	70 %	7 %	8 %	10 %
	0.75	5 %	52 %	6 %	7 %	7 %
	1.00	4 %	27 %	5 %	4 %	4 %
manual	0.00	5 %	77 %	8 %	12 %	16 %
	0.25	5 %	75 %	8 %	9 %	12 %
	0.50	6 %	56 %	8 %	9 %	10 %
	0.75	7 %	42 %	8 %	7 %	11 %
	1.00	9 %	27 %	9 %	13 %	14 %
Average		5 %	58 %	7 %	8 %	10 %

Table 6.7: Effect of varying T_C over T_Q , using F1 score. The measures are averaged over all $T_S \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

F1 Score		T_C					Average
Tagging	T_Q	0.00	0.25	0.50	0.75	1.00	
auto	0.00	8 %	8 %	8 %	8 %	4 %	7 %
	0.25	8 %	8 %	7 %	7 %	4 %	7 %
	0.50	8 %	8 %	7 %	6 %	4 %	7 %
	0.75	8 %	7 %	7 %	6 %	5 %	7 %
	1.00	6 %	6 %	5 %	3 %	7 %	5 %
manual	0.00	8 %	8 %	8 %	5 %	2 %	6 %
	0.25	8 %	8 %	8 %	6 %	2 %	6 %
	0.50	8 %	8 %	7 %	5 %	3 %	6 %
	0.75	8 %	8 %	8 %	12 %	16 %	11 %
	1.00	7 %	7 %	11 %	13 %	22 %	12 %
Average		7 %	7 %	8 %	7 %	7 %	7 %

Picking threshold values for DataOntoSearch version 2

The previous subsection showed how important it is to pick good threshold values for DataOntoSearch. Even though the tables from that section could be used to simply pick the best values available, this would risk giving DataOntoSearch an unfair advantage. After all, none of the other two engines are given the opportunity to adjust their inner workings to the queries at hand.

To make this more fair, around 15% of the queries, selected randomly, are set aside and are only used to adjust DataOntoSearch’s thresholds. The remaining 85% of the queries are used for evaluation only. This reflects the method often used in classification, where a certain number of tasks are used as training set and the rest as validation and testing set.

The reliability of this method is hindered by the fact that only one split between training and testing set is done. The particular numbers are therefore sensitive to what queries end up in what set, since some training queries can be more representative of the testing set than others. This effect is reinforced by the low number of queries in general. That said, this reflects how the system might be set up in the wild, where the threshold values are likely “set once and forgotten.” Though it is not perfect, I consider this more representative of DataOntoSearch’s performance than the aforementioned approach of picking values using *all* queries, adding some imperfection to the mix.

Based on the queries in the training set, the thresholds for DataOntoSearch version 2 were set as follows: $T_S = 0.25$, $T_Q = 0.75$, $T_C = 1.00$. The results reported for DataOntoSearch version 2 in the following were gathered using those thresholds.

As can be seen in Table 6.9-6.12, the combination $T_S = 0.25$, $T_Q = 1.00$, $T_C = 1.00$ yields better results when using all queries, than the combination chosen based on the training set. To see what difference this makes for the measurements, a comparison between the two threshold combinations is made in Table 6.13 on page 96, using all queries and not just the testing set.

The threshold variables were not new to DataOntoSearch version 2. T_S was already present in DataOntoSearch version 1, but it was hard-coded in the code. Though [25] claims that T_C was also implemented, its implementation could not be found in the source code I had access to. It was likely implemented in some code that was never published to GitHub. Due to these issues, the threshold values for DataOntoSearch version 1 are effectively set to $T_S = 0.75$, $T_Q = 0.00$, $T_C = 0.00$. This is likely to give version 1 a disadvantage compared to version 2, but it also reflects the state of the system.

Note that the effect of the threshold values has changed between the versions, due to the implementation changes made to the query processing. It therefore does not make sense to set the threshold to be equal on the two versions. I experimented with giving version 1 the same threshold values as version 2, but the system ended up not returning any datasets.

Evaluation with concept-based relevance assessment

The results from evaluating the open data search engines DataOntoSearch version 2, Google Dataset Search and DataOntoSearch version 1 are presented here.

To decide what datasets are relevant to each query, we use the manual associations made between datasets and concepts. Each query has been manually associated with con-

cepts, and only datasets that are associated with the same concepts are considered relevant for that query. The details of how the queries were gathered and associated with concepts are covered in Section 6.1.3.

Precision The reported precision for the three systems is shown in Table 6.14 on page 96. DataOntoSearch version 1 performs the worst, with no big difference between the automatic and manual tagging. Google Dataset Search performs reasonably well, though there are some queries it is unable to handle. DataOntoSearch version 2 varies greatly between the automatic and the manual tagging, with the manual tagging being much more precise, even beating Google.

Recall The recall scores found are presented in Table 6.15 on page 96. The automatic tagging of version 1 consistently returns a lot of the relevant datasets, though as we saw in the precision scores, it also returns many irrelevant datasets. The other configurations and systems perform about the same, though version 2 with the manual tagging returns about all relevant datasets for a couple of the queries, granting it about half the recall score of version 1 with automatic tagging.

F1 score Table 6.16 on page 97 shows the F1 scores, which present a unified view of the precision and recall measures. Even though version 1 had a pretty high measured recall, its low precision keeps its F1 score down. Google, on the other hand, is kept down by their relatively low recall. Version 2 with automatic tagging does not perform much better than version 1, but the manual tagging scores very high for some queries, with some queries even giving perfect results, i.e. all relevant datasets being retrieved and nothing else. This grants it the highest average, though it seems to be very dependent on the type of queries.

R-Precision Moving on to the ranking-based algorithms, the R-Precision measurements are reported in Table 6.17 on page 97. For DataOntoSearch version 1, curiously enough, the automatic tagging performs better than the manual tagging. Google performs somewhat better than version 1, but is once again beat by version 2 with manual tagging. The same queries we saw perform well with the previous measures also do well here, reinforcing the engine's sensitivity to the type of queries.

Mean Average Precision (MAP) The mean average precision is at display in Table 6.18 on page 98. It is not so different from the results of R-Precision. Though there are some changes here and there, the overall trend is the same.

Summary The different systems' averages over all the queries are presented in Table 6.19 on page 98. DataOntoSearch version 2 with manual tagging performs the best overall, with Google Data Set Search being the next best. Though as we saw, how the systems perform depend very much on the queries used.

Evaluation with task-based relevance assessment

Though we have just seen how the systems perform in a traditional evaluation, the method used has some limitations. Specifically, the relevance assessment uses information which is only available to DataOntoSearch’s manual tagging, and not known to the automatic tagging or Google. At the same time, we already know what queries users would make when put in front of the two systems – their queries during the pre-study were recorded. Using these queries, we can look at how well the systems under test would respond to the test users’ needs, by using the queries test users made, and marking the dataset that would solve their task as the relevant dataset.

Table 6.20 on page 98 shows the result of this experiment, using mean average precision (MAP). Aside from the queries being different, it shows how version 2 of DataOntoSearch fares much better than version 1. Google performs just as well as version 2 when the rest stops are queried more or less directly, but it is unable to find it when it is only asked about indirectly. That said, DataOntoSearch does not fare much better in those cases.

Google’s performance is greatly hindered by the aforementioned bug, since it is unable to retrieve the relevant dataset for three of the queries.

6.3.3 Performance

I tested the performance, as in execution time, using the `perf stat` tool. I used the multisearch utility of DataOntoSearch, which runs many queries after each other, with varying threshold values. It was run with two rather lengthy queries:

- highway new york rural
- underground railway position

Due to the amount of threshold values specified, a total of 500 searches were made during this execution.

The first time a query is run, WordNet is loaded into memory. This is done lazily, i.e. WordNet does not load before it is needed, which incidentally is when similarities are calculated between concepts and the query. It therefore disturbs the information about how long the different phases of the search take, and penalizes systems which cannot run multiple queries in one execution, since they will need to load WordNet for every query. This is why it is not possible to compare the performance between version 1 and version 2 of DataOntoSearch.

Some performance statistics of DataOntoSearch version 1 are reported in [25]. I only report the runtime of queries in version 2 here, since I presume that the processing time users will actually need to wait for is the most important. The offline processes are therefore not as interesting.

The computer used for this testing has an Intel Core i5-g200U CPU, capable of running 2.30GHz with 4 processors, running Fedora 30. The hard disk is a Solid State Drive (SSD) and it has 7.7GiB of memory. Other processes took up about half of the processing power available, with a heavy memory load.

The statistics reported by `perf stat`, averaged over all searches performed:

- **Time spent in total:** 5.626 seconds. This also includes time in which other processes occupied the processor.
- **Time spent running in userland:** 5.575 seconds. This essentially means time spent performing calculations and doing things.
- **Time spent running in the kernel:** 0.170 seconds. This is the time the process spent waiting for the kernel to return, typically while the kernel handles network communication or disk reads.
- **Mispredicted branches:** 1.16 %.

The time spent running in userland and kernel add up to more than the total time, likely due to some small amount of parallelism that is being used in one of the libraries. The former measures count time per processor, so when two processors are involved, the time is counted twice.

When excluding the first query run, the time spent for the different parts of the query are shown in Table 6.21 on page 98. The first query took 7.399 seconds to run.

As shown, the first part of the process is where most of the time is spent. From my time experimenting with the system, I noticed that this portion did not take so much time before the CCSM was taken into use. In an attempt to reduce the time spent, I used a profiling tool for Python to see which functions took much time, and the function involved with applying the CCSM to the QCSV was indeed the one where the most time was spent. Finding a way of applying the CCSM to QCSV inside NumPy, instead of in Python code, will likely reduce this time drastically.

It should be noted that the part of the process called *Putting together result* is not relevant in this case, because the dataset metadata is not gathered when running multiple queries, since it is not used. This process involves looking up the dataset in the dataset graph and creating a structure of metadata. It escalates linearly with the number of datasets retrieved, and is therefore sensitive to how the thresholds are set. From a repeated test with a single query, it took around 0.060 and 0.065 seconds to do this when fetching the dataset metadata and 3.224 and 3.588 seconds when also fetching the concepts most similar to each dataset. This was for a query that returned 282 datasets.

Table 6.8: Effect of varying T_C over T_Q , using MAP. The measures are averaged over all $T_S \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$

MAP		T_C					Average
Tagging	T_Q	0.00	0.25	0.50	0.75	1.00	
auto	0.00	11 %	11 %	11 %	6 %	3 %	8 %
	0.25	11 %	11 %	10 %	6 %	3 %	8 %
	0.50	11 %	11 %	9 %	11 %	4 %	9 %
	0.75	11 %	10 %	11 %	9 %	5 %	9 %
	1.00	7 %	7 %	6 %	4 %	7 %	6 %
manual	0.00	10 %	9 %	7 %	6 %	4 %	7 %
	0.25	10 %	9 %	7 %	6 %	5 %	7 %
	0.50	10 %	9 %	7 %	6 %	6 %	8 %
	0.75	18 %	9 %	8 %	16 %	23 %	15 %
	1.00	29 %	24 %	22 %	20 %	32 %	25 %
Average		13 %	11 %	10 %	9 %	9 %	10 %

Table 6.9: Effect of varying T_S over combinations of T_Q and T_C for automatic tagging, measured using F1 score

F1 score, auto		T_S					Average
T_Q	T_C	0.00	0.25	0.50	0.75	1.00	
0.00	0.00	9 %	9 %	9 %	9 %	0 %	8 %
	0.25	9 %	9 %	9 %	9 %	0 %	8 %
	0.50	9 %	9 %	9 %	10 %	0 %	8 %
	0.75	9 %	9 %	10 %	9 %	0 %	8 %
	1.00	9 %	11 %	0 %	0 %	0 %	4 %
0.25	0.00	9 %	9 %	9 %	9 %	0 %	8 %
	0.25	9 %	9 %	9 %	9 %	0 %	8 %
	0.50	9 %	9 %	9 %	7 %	0 %	7 %
	0.75	9 %	9 %	10 %	7 %	0 %	7 %
	1.00	9 %	12 %	0 %	0 %	0 %	4 %
0.50	0.00	9 %	9 %	9 %	9 %	0 %	8 %
	0.25	9 %	9 %	9 %	9 %	0 %	8 %
	0.50	9 %	9 %	9 %	7 %	0 %	7 %
	0.75	9 %	7 %	8 %	7 %	0 %	6 %
	1.00	9 %	12 %	0 %	0 %	0 %	4 %
0.75	0.00	9 %	9 %	9 %	9 %	0 %	8 %
	0.25	9 %	9 %	9 %	7 %	0 %	7 %
	0.50	9 %	9 %	7 %	11 %	0 %	7 %
	0.75	9 %	8 %	9 %	4 %	0 %	6 %
	1.00	9 %	14 %	3 %	0 %	0 %	5 %
1.00	0.00	9 %	6 %	6 %	6 %	0 %	6 %
	0.25	9 %	6 %	6 %	6 %	0 %	6 %
	0.50	9 %	7 %	9 %	1 %	0 %	5 %
	0.75	9 %	5 %	1 %	0 %	0 %	3 %
	1.00	9 %	15 %	9 %	0 %	0 %	7 %
Average		9 %	9 %	7 %	6 %	0 %	6 %

Table 6.10: Effect of varying T_S over combinations of T_Q and T_C for manual tagging, measured using F1 score

F1 score, manual		T_S					Average
T_Q	T_C	0.00	0.25	0.50	0.75	1.00	
0.00	0.00	10 %	10 %	10 %	10 %	0 %	8 %
	0.25	10 %	10 %	10 %	10 %	0 %	8 %
	0.50	10 %	10 %	11 %	8 %	0 %	8 %
	0.75	10 %	11 %	6 %	0 %	0 %	5 %
	1.00	10 %	0 %	0 %	0 %	0 %	2 %
0.25	0.00	10 %	10 %	10 %	10 %	0 %	8 %
	0.25	10 %	10 %	10 %	10 %	0 %	8 %
	0.50	10 %	10 %	12 %	7 %	0 %	8 %
	0.75	10 %	12 %	7 %	0 %	0 %	6 %
	1.00	10 %	1 %	0 %	0 %	0 %	2 %
0.50	0.00	10 %	10 %	10 %	10 %	0 %	8 %
	0.25	10 %	10 %	10 %	10 %	0 %	8 %
	0.50	10 %	10 %	12 %	5 %	0 %	7 %
	0.75	10 %	10 %	5 %	0 %	0 %	5 %
	1.00	10 %	6 %	0 %	0 %	0 %	3 %
0.75	0.00	10 %	10 %	10 %	10 %	3 %	8 %
	0.25	10 %	10 %	10 %	12 %	0 %	8 %
	0.50	10 %	12 %	16 %	5 %	0 %	8 %
	0.75	10 %	29 %	19 %	5 %	0 %	12 %
	1.00	10 %	40 %	18 %	9 %	3 %	16 %
1.00	0.00	10 %	7 %	7 %	7 %	5 %	7 %
	0.25	10 %	7 %	7 %	9 %	5 %	7 %
	0.50	10 %	9 %	14 %	17 %	5 %	11 %
	0.75	10 %	18 %	19 %	16 %	0 %	13 %
	1.00	10 %	55 %	36 %	5 %	5 %	22 %
Average		10 %	13 %	11 %	7 %	1 %	8 %

Table 6.11: Effect of varying T_S over combinations of T_Q and T_C for automatic tagging, measured in MAP

MAP, auto		T_S					Average
T_Q	T_C	0.00	0.25	0.50	0.75	1.00	
0.00	0.00	13 %	13 %	13 %	13 %	0 %	11 %
	0.25	13 %	13 %	13 %	13 %	0 %	11 %
	0.50	14 %	14 %	14 %	14 %	0 %	11 %
	0.75	9 %	8 %	8 %	6 %	0 %	6 %
	1.00	9 %	5 %	0 %	0 %	0 %	3 %
0.25	0.00	13 %	13 %	13 %	13 %	0 %	11 %
	0.25	14 %	14 %	14 %	14 %	0 %	11 %
	0.50	14 %	14 %	14 %	11 %	0 %	10 %
	0.75	9 %	9 %	9 %	4 %	0 %	6 %
	1.00	10 %	6 %	0 %	0 %	0 %	3 %
0.50	0.00	13 %	13 %	13 %	13 %	0 %	11 %
	0.25	13 %	13 %	13 %	13 %	0 %	11 %
	0.50	13 %	13 %	11 %	10 %	0 %	9 %
	0.75	19 %	15 %	15 %	5 %	0 %	11 %
	1.00	12 %	6 %	0 %	0 %	0 %	4 %
0.75	0.00	14 %	14 %	14 %	14 %	0 %	11 %
	0.25	14 %	14 %	14 %	8 %	0 %	10 %
	0.50	15 %	15 %	13 %	11 %	0 %	11 %
	0.75	19 %	16 %	11 %	1 %	0 %	9 %
	1.00	16 %	9 %	2 %	0 %	0 %	5 %
1.00	0.00	11 %	8 %	8 %	8 %	0 %	7 %
	0.25	10 %	8 %	8 %	7 %	0 %	7 %
	0.50	12 %	10 %	7 %	0 %	0 %	6 %
	0.75	13 %	7 %	1 %	0 %	0 %	4 %
	1.00	19 %	12 %	5 %	0 %	0 %	7 %
Average		13 %	11 %	9 %	7 %	0 %	8 %

Table 6.12: Effect of varying T_S over combinations of T_Q and T_C for manual tagging, measured in MAP

MAP, manual		T_S					Average
T_Q	T_C	0.00	0.25	0.50	0.75	1.00	
0.00	0.00	13 %	13 %	13 %	13 %	0 %	10 %
	0.25	11 %	11 %	11 %	11 %	0 %	9 %
	0.50	11 %	11 %	11 %	4 %	0 %	7 %
	0.75	13 %	12 %	3 %	0 %	0 %	6 %
	1.00	22 %	0 %	0 %	0 %	0 %	4 %
0.25	0.00	13 %	13 %	13 %	13 %	0 %	10 %
	0.25	11 %	11 %	11 %	11 %	0 %	9 %
	0.50	11 %	11 %	11 %	4 %	0 %	7 %
	0.75	14 %	12 %	4 %	0 %	0 %	6 %
	1.00	22 %	0 %	0 %	0 %	0 %	5 %
0.50	0.00	13 %	13 %	13 %	13 %	0 %	10 %
	0.25	11 %	11 %	11 %	11 %	0 %	9 %
	0.50	11 %	11 %	11 %	2 %	0 %	7 %
	0.75	17 %	11 %	3 %	0 %	0 %	6 %
	1.00	26 %	5 %	0 %	0 %	0 %	6 %
0.75	0.00	22 %	22 %	22 %	22 %	2 %	18 %
	0.25	12 %	12 %	12 %	11 %	0 %	9 %
	0.50	14 %	14 %	13 %	3 %	0 %	8 %
	0.75	33 %	30 %	14 %	4 %	0 %	16 %
	1.00	52 %	36 %	17 %	7 %	2 %	23 %
1.00	0.00	36 %	34 %	34 %	34 %	4 %	29 %
	0.25	31 %	29 %	29 %	29 %	4 %	24 %
	0.50	31 %	28 %	28 %	19 %	4 %	22 %
	0.75	32 %	27 %	24 %	18 %	0 %	20 %
	1.00	59 %	56 %	34 %	4 %	4 %	32 %
Average		22 %	17 %	14 %	9 %	1 %	13 %

Table 6.13: Summary of measurements between the two threshold combinations, using all queries. Google is included for comparison.

Measure	v2, best		v2, chosen		Google
	auto	manual	auto	manual	
Precision	13 %	54 %	13 %	42 %	26 %
Recall	23 %	58 %	27 %	45 %	18 %
F1 score	15 %	55 %	14 %	40 %	17 %
R-Precision	13 %	56 %	9 %	35 %	18 %
Mean Average Precision (MAP)	12 %	56 %	9 %	36 %	18 %

Table 6.14: Precision for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Query	v2		Google	v1		Average
	auto	manual		auto	manual	
bike	31 %	0 %	100 %	12 %	7 %	30 %
location transportation	6 %	0 %	40 %	7 %	24 %	15 %
map	5 %	100 %	0 %	5 %	2 %	22 %
messaging	0 %	100 %	0 %	0 %	0 %	20 %
new york state rest stop	0 %	67 %	50 %	1 %	2 %	24 %
nyc subway	33 %	60 %	0 %	2 %	0 %	19 %
rest area	0 %	67 %	67 %	1 %	3 %	27 %
road traffic management	6 %	14 %	0 %	3 %	0 %	5 %
site:data.cityofnewyork.us subway stations	4 %	0 %	0 %	1 %	0 %	1 %
statistics transportation	48 %	100 %	100 %	38 %	46 %	66 %
stopping place	0 %	0 %	0 %	1 %	0 %	0 %
subway stations	5 %	0 %	0 %	1 %	0 %	1 %
walking	3 %	0 %	0 %	0 %	0 %	1 %
Average	11 %	39 %	27 %	5 %	6 %	18 %

Table 6.15: Recall for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Query	v2		Google	v1		Average
	auto	tagged		auto	tagged	
bike	31 %	0 %	15 %	85 %	15 %	29 %
location transportation	15 %	0 %	20 %	100 %	65 %	40 %
map	15 %	100 %	0 %	100 %	8 %	45 %
messaging	0 %	100 %	0 %	0 %	0 %	20 %
new york state rest stop	0 %	100 %	50 %	100 %	50 %	60 %
nyc subway	67 %	100 %	0 %	100 %	0 %	53 %
rest area	0 %	100 %	100 %	100 %	100 %	80 %
road traffic management	20 %	10 %	0 %	100 %	0 %	26 %
site:data.cityofnewyork.us subway stations	50 %	0 %	0 %	100 %	0 %	30 %
statistics transportation	10 %	4 %	5 %	100 %	21 %	28 %
stopping place	0 %	0 %	0 %	100 %	0 %	20 %
subway stations	50 %	0 %	0 %	100 %	0 %	30 %
walking	13 %	0 %	0 %	0 %	0 %	3 %
Average	21 %	40 %	15 %	83 %	20 %	36 %

Table 6.16: F1 score for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Query	v2		Google	v1		Average
	auto	tagged		auto	tagged	
bike	31 %	0 %	27 %	21 %	10 %	18 %
location transportation	8 %	0 %	27 %	13 %	35 %	17 %
map	7 %	100 %	0 %	9 %	3 %	24 %
messaging	0 %	100 %	0 %	0 %	0 %	20 %
new york state rest stop	0 %	80 %	50 %	1 %	3 %	27 %
nyc subway	44 %	75 %	0 %	4 %	0 %	25 %
rest area	0 %	80 %	80 %	1 %	5 %	33 %
road traffic management	10 %	12 %	0 %	7 %	0 %	6 %
site:data.cityofnewyork.us subway stations	8 %	0 %	0 %	1 %	0 %	2 %
statistics transportation	17 %	7 %	10 %	55 %	29 %	24 %
stopping place	0 %	0 %	0 %	2 %	0 %	0 %
subway stations	9 %	0 %	0 %	1 %	0 %	2 %
walking	5 %	0 %	0 %	0 %	0 %	1 %
Average	11 %	35 %	15 %	9 %	6 %	15 %

Table 6.17: R-Precision for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Query	v2		Google	v1		Average
	auto	manual		auto	manual	
bike	31 %	0 %	15 %	15 %	15 %	15 %
location transportation	15 %	0 %	20 %	10 %	5 %	10 %
map	15 %	100 %	0 %	15 %	8 %	28 %
messaging	0 %	100 %	0 %	0 %	0 %	20 %
new york state rest stop	0 %	50 %	50 %	0 %	0 %	20 %
nyc subway	0 %	100 %	0 %	67 %	0 %	33 %
rest area	0 %	50 %	100 %	0 %	0 %	30 %
road traffic management	10 %	10 %	0 %	0 %	0 %	4 %
site:data.cityofnewyork.us subway stations	0 %	0 %	0 %	0 %	0 %	0 %
statistics transportation	10 %	4 %	5 %	28 %	21 %	14 %
stopping place	0 %	0 %	0 %	0 %	0 %	0 %
subway stations	0 %	0 %	0 %	0 %	0 %	0 %
walking	0 %	0 %	0 %	0 %	0 %	0 %
Average	6 %	32 %	15 %	10 %	4 %	13 %

Table 6.18: Mean Average Precision (MAP) for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Query	v2		Google	v1		Average
	auto	manual		auto	manual	
bike	10 %	0 %	15 %	17 %	5 %	9 %
location transportation	4 %	0 %	18 %	12 %	11 %	9 %
map	15 %	100 %	0 %	7 %	1 %	25 %
messaging	0 %	100 %	0 %	0 %	0 %	20 %
new york state rest stop	0 %	58 %	50 %	4 %	1 %	23 %
nyc subway	19 %	100 %	0 %	58 %	0 %	35 %
rest area	0 %	58 %	100 %	5 %	2 %	33 %
road traffic management	3 %	1 %	0 %	6 %	0 %	2 %
site:data.cityofnewyork.us subway stations	10 %	0 %	0 %	7 %	0 %	3 %
statistics transportation	4 %	4 %	5 %	32 %	13 %	12 %
stopping place	0 %	0 %	0 %	1 %	0 %	0 %
subway stations	3 %	0 %	0 %	11 %	0 %	3 %
walking	1 %	0 %	0 %	0 %	0 %	0 %
Average	5 %	32 %	15 %	12 %	3 %	13 %

Table 6.19: Summary of measurements for DataOntoSearch v2, Google Dataset Search and DataOntoSearch v1.

Measure	v2		Google	v1		Average
	auto	manual		auto	manual	
Precision	11 %	39 %	27 %	5 %	6 %	18 %
Recall	21 %	40 %	15 %	83 %	20 %	36 %
F1 score	11 %	35 %	15 %	9 %	6 %	15 %
R-Precision	6 %	32 %	15 %	10 %	4 %	13 %
Mean Average Precision (MAP)	5 %	32 %	15 %	12 %	3 %	13 %

Table 6.20: Mean Average Precision (MAP) using task-based relevance assessment

Query	v2		Google	v1		Average
	auto	manual		auto	manual	
new york public facilities	0 %	0 %	0 %	0 %	0 %	0 %
new york road	0 %	0 %	0 %	0 %	0 %	0 %
new york state highway public facilities	0 %	0 %	0 %	0 %	2 %	0 %
new york state rest stop	0 %	50 %	0 %	0 %	2 %	10 %
nyc subway	0 %	100 %	0 %	2 %	0 %	20 %
nyc subway station	5 %	0 %	0 %	2 %	0 %	2 %
rest area	0 %	50 %	50 %	0 %	2 %	20 %
rest stops	0 %	50 %	50 %	0 %	2 %	20 %
site:data.cityofnewyork.us subway stations	20 %	0 %	0 %	2 %	0 %	4 %
Average	3 %	28 %	11 %	1 %	1 %	9 %

Table 6.21: The time spent on the different parts of the query process. Measured in real time.

Part of process	Average time (seconds)
Associating query with concepts	4.506
Finding the most similar concepts	0.000
Comparing QDSV with each row of the DCSV	0.017
Putting together result	0.001
<i>Start to end</i>	4.525

Discussion

In Chapter 5, we saw how DataOntoSearch version 1 fared in the hands of test users in a usability test. Along with this, I put in an effort to make the system runnable and increase its modifiability. Then, in Chapter 6, I improved on the system, both by making it available to CKAN users and by improving its search algorithm. I evaluated the system along with version 1 and Google Dataset Search, using a number of different system-oriented evaluation methods.

One question remains, though: What does it all mean? What conclusions can we draw from all this? How do we answer the research questions?

Recall the two research questions presented in Section 4.2:

RQ1: What do users think of DataOntoSearch version 1? What main problems are there?

RQ2: How can we address the identified problems when creating version 2?

First, RQ1 is discussed in Section 7.1. RQ2 is discussed through the lens of the CKAN plugin in Section 7.2, the code quality improvements in Section 7.3, the systematic evaluation in Section 7.4 and run-time performance in Section 7.5. In section 7.6, DataOntoSearch’s approach is compared to that of Google Dataset Search and the spatio-temporal search. Finally, in Section 7.7, I list up the directions that future work can take.

7.1 The usability test

When looking at the results of the usability test (Table 5.2 on page 58), it is not difficult to see which system could be considered the “winner.” Where DataOntoSearch was given an average rating of 3.75 by the four test users, Google Dataset Search was given 6.25. The latter system actually managed to answer the users’ needs, while the search results of DataOntoSearch almost seemed random and not connected to the query. In fact, some datasets always hugged the top positions, even if the user searched for something different.

That said, the test tasks were formulated so that they did not give away the name of the dataset. This turned out to be a good decision, since the users often used terminology

from the task directly. Especially for the task of finding rest areas, most test users were unfamiliar with the term “rest area,” performing searches like “new york state rest stop” (a direct translation of the Norwegian term “rasteplass”), and “highway new york rural” (using keywords from the task). For those queries, Google Dataset Search shows the same weaknesses that CKAN’s built-in search also shows, in that “rest stop” does not give the same results as “rest area,” even though they are intuitively related. This goes to show that the ontology-based semantic approach may still hold some promise, it just had not come to fruition yet with DataOntoSearch version 1.

Users were already familiar with Google, though they had not heard about Google Dataset Search before. Its familiarity, both in brand and function, likely gave test users a better first-impression than DataOntoSearch’s unstyled interface. One user jokingly commented something along the lines of “ah, nice to see Google’s famous autocomplete again.”

There was no uniform opinion on the layout, where the results are listed on the left with a main window presenting the selected dataset. Some preferred DataOntoSearch’s traditional presentation which gave much information right away, while others liked the potential for seeing additional details without leaving the search results page.

Since the users were not given the exact words of the dataset, they had to perform an assessment themselves of what datasets were relevant or not. This involved reading the title and description, and clicking to see how the dataset is presented at its source, e.g. the open data portals of NYC and NYS. However, they did not just open any dataset to look at. The open data portals are quite slow to respond, and they did not always answer the user’s question, likely stopping the users from wanting to check every dataset.

One user commented that they really liked how the New York State open data portal showed a preview of the dataset, since that could confirm that the dataset contained the information the user sought. The same user wrote in the comparative questionnaire that “A data preview would also be appreciated” for both search systems, and commented that in a “real situation,” they might just try to go “directly to data.gov, or data.norge,” preferring browsing over searching for such exploratory tasks. It should be noted that this user is knowledgeable about how transport systems are usually organized, and so would have a better chance of knowing who the dataset publisher is than people less familiar with those structures.

A curious part of the usability test setup, was the fact that I included a task for which no relevant dataset exists. This is not something that I have seen described anywhere, but I decided to include it to avoid users trying to search for much longer than they normally would. I cannot say that I observed the desired effect. The longest time spent on a single task was almost 10 minutes before successfully finding the dataset. It seems that the users’ desire to succeed with the tasks outweighed their perceived chance that there are no relevant datasets to be found. The difference in time spent on Google Dataset Search and DataOntoSearch is not very big, and may have been influenced by what tasks the users were given on what systems. The time difference between tasks is greater than the time difference between systems.

Another explanation may be that the tasks seemed very simple to do. One user was baffled by what datasets are available, saying that “there are many datasets, like number of transported railcars through New York and New Jersey every year since 2000. . . but not exactly relevant,” later noting that they could find the bridge conditions of individual

bridges, but not something as “simple” as the Amtrak train departure times.

The usability test showed that developers, though curious about the semantic search engine, approached it just like they would approach any other search. Although they were given a manual they could read, most did not take the time to read it, opting instead to just play with the system directly. This likely reflects how users would approach the search *in the wild*, though, and shows that it is difficult to make users change their behaviour. I speculate that if you want to create a semantic search which places a burden on the user in how they formulate the query, you might want to re-think and adopt a more guided approach, so the user is taken out of their “Google” mindset. If you just provide a big search box, users are likely to engage with it just like they engage with any other search. Why shouldn't they?

The main takeaways from the evaluation of DataOntoSearch version 1 and Google Dataset Search, answering RQ1, are:

- DataOntoSearch version 1 left a bad impression, and did not help the users solve their tasks.
- Users liked Google Dataset Search, though its unique layout was divisive.
- Dataset preview gave users a lot more than just the dataset description alone, and were appreciated.
- Some users may prefer to browse categories and publishers, rather than rely on search.
- Including tasks which cannot be solved did not seem to have any effect, though I have not run any evaluation *without* the task in question, so I cannot really conclude one way or another. It may just be enough to include an upper time limit.
- You cannot assume that the users will use the terminology that is used by the dataset publisher, even for simple topics like *rest area* or *subway*.
- Users were surprised about what datasets were available and what datasets were not available.
- If you give users a search box, then they will treat it just like they would treat any other search, even if you advertise it as a different kind of search.

The users tested with here were, for the most part, not overly familiar with the transport domain. The way they approached the systems is therefore likely to be different from how domain experts would approach them, since the latter should be able to more consistently use the same words as the dataset publishers and the concepts in our domain-specific ontology. That said, one of the motivations for experimenting with an ontology-based semantic search approach were to make the existing datasets available to a broader audience than just the domain experts.

7.1.1 Limitations

There are some limitations to the usability test. First of all, different interfaces are used for the different search engines. This means that the results tell us about the entire experience, and not just about the differences in the search algorithms. I wanted to know how the two different search interfaces made for different user experiences anyway, so this was an accepted limitation.

A bigger problem is how the test users were recruited among my own friends and acquaintances. There is a chance that some may have been more tenacious than they otherwise would have been, because they were testing with someone they knew. I did emphasize that I was interested in learning how the *system* fared, not the user, and that any failings would be attributed to the system.

Another problem with selecting users like this is a potential for lack of diversity in test users, meaning that my findings may not be representable of all the different kinds of people. Since the usability test was formative and not summative, I had accepted from the start that the results did not need to be generalized to a bigger population, which is also why I deemed four test users to be adequate. This does pose a restriction on what can be concluded based on the usability test, as was explained when the results were presented in Chapter 5. That said, a trade-off had to be made, and the existing literature on usability testing suggests that you will see diminishing returns for qualitative usability tests when you add users beyond the fifth [35]. We got to know that DataOntoSearch was performing poorly after testing with just four users, and I would probably not have had any time left to improve the system if I were to evaluate with more users.

7.2 The CKAN plugin

One of the main contributions of this thesis is the integration of DataOntoSearch into CKAN. It provides a new way of accessing the search and manual tagging functionality of DataOntoSearch, without users even realizing that it is a separate system. This is vital first step in making DataOntoSearch a usable system that can actually be used and help users find the datasets they want. This also comes with improvements to the overall user experience, since CKAN's design is more stylized and is probably able to evoke more authority than the bare-bones design of DataOntoSearch's web user interface, thus helping answer RQ2.

Developing plugins for CKAN was surprisingly hard and complex. There is no documentation on the classes and functions available, with very few exceptions. In order to understand it all, you must dive into the code itself and read the documentation embedded there. I did not expect that plugin authors had to read the source of CKAN itself, and while the documentation for plugin authors seems adequate at first, it e.g. does not mention what route name you must use to create a hyperlink to a dataset. There is also very little guidance, so I almost implemented routes using the old mechanism that CKAN is transitioning from, and not the new mechanism.

Despite these problems, actually writing the extension was not so difficult. The challenge lied in knowing what to write and what to use, and not in the actual engineering itself.

Now that one integration has been made for DataOntoSearch, it is not hard to make integrations for more systems. A big part of the work has already been done, since DataOntoSearch now has APIs capable of handling semantic search and manual tagging. These APIs are not restricted to use by CKAN, and give DataOntoSearch added flexibility and longevity since it is able to survive transitions between dataset archival solutions.

7.2.1 Limitations

Although I think the CKAN integration improves the usability of DataOntoSearch, there has not been enough time available to conduct a second usability test to confirm the increase in the system's ability to fetch relevant datasets or the improvements made by adapting it for CKAN. Once more improvements are made to the CKAN integration, it would be interesting to know how users feel about using the user interface, compared to the built-in interface of DataOntoSearch. It may also be revelatory to check how the improved semantic search fares compared to Google Dataset Search and CKAN's native search.

Since the CKAN integration does not change anything about the search procedure, no changes can be observed from its implementation in the evaluation results of RQ2. Therefore, there is no evaluation to back up my claims about the CKAN integration improving the user experience of DataOntoSearch, other than my own anecdotal experience with trying out the system for myself. There simply was not enough time to perform a second usability test when so much time had already gone into preparing the first one, including the refactoring work.

7.3 Code quality

Converting DataOntoSearch from being a prototype to being a system others can adopt and use, has generally been a recurring theme in my work. A significant portion of the time spent preparing the usability tests were spent making the system easier to reason about and modify, and though this isn't something that can be easily measured, it still remains an important part of this thesis. Especially considering how, in academia, it seems like a rarity to have projects that amount to anything more than just qualifying the students for a certain grade. Hopefully, DataOntoSearch version 2 can help provide any developers new to the system with a good introduction and code which they can build on. Only then can the system evolve into a better version of itself, instead of new developers taking one glance at the project, shrugging and starting to write a new version from scratch.

That said, I might have been better off starting from scratch myself, and just copy in relevant parts from the existing system. The architecture and storage solutions were mostly inherited from Hagelien's system, and embed decisions that made sense when prototyping and experimenting, but which may not make much sense for the system the way it has grown. For example, the search and construction of the matrices is located entirely in a class named `OpenDataSemanticFramework`. It is not entirely clear what the class or its instances represent, yet I have kept it to avoid having to spend effort rewriting it. Instead, I have written a *new* class called `ODSFLoader`, which simply is responsible for creating instances of `OpenDataSemanticFramework` and initialize them on demand, and keep a copy of instances so they can be re-used later. Though it made sense to do at the time, to avoid

dealing with the existing infrastructure, this is probably just a symptom of architectural problems with DataOntoSearch.

The inherited storage solution is also not fit for purpose. It requires using MongoDB, which has adopted a license that is *not* considered a “Free Software License” and the project is therefore thrown out of e.g. Fedora’s repositories [30]. Its document-centric approach, while convenient for an experimentation phase where you don’t know what your schema will be, does not really do DataOntoSearch any favours. Even though the RDF graphs are stored using JSON-LD, which you would think would make them accessible to MongoDB’s search functions since it uses JSON itself, the graphs are encoded as *binary* JSON and thus completely inaccessible to MongoDB. There exist storage solutions out there specifically targeting RDF graphs, which should be investigated.

Still, by converting DataOntoSearch from being a bunch of Jupyter Notebooks to consisting of Python scripts, and by drastically increasing the project’s quality, the system stands a much greater chance of being useful in the future.

7.4 Search quality and evaluation

During my work with DataOntoSearch version 2, I managed to sneak in a couple changes to the search process to help address RQ2. Though the changes were small and innocent on their own, they have collectively increased the quality of the search results. WordNet is also utilized to a greater extent, and all the concepts of the ontology can now actually be associated with the user’s query.

The usability test showed that DataOntoSearch version 1 returned what felt like random datasets. The results from the final evaluation confirms that the performance of version 1 with manual linking is not good, and the automatic linking, curiously, performed a little better than the manual linking, though it returns way too many datasets, as can be seen from the high recall value.

Regarding the thresholds, you can see that T_S very clearly is the cut-off point for the search results. In general, you can achieve a perfect recall score by returning all datasets, but then your precision score is not going to be so great. On the other hand, you can retrieve very few datasets and have a high precision, but low recall. Achieving a good F1 score requires that you perform well with both precision and recall. T_S is therefore important for achieving a good performance with the unranked measures. Setting it to 0.00 simply returns all datasets, so the scores are the same independent of the other thresholds. For the other thresholds, T_S needs to be set so that not too many irrelevant datasets are “above the fold,” while still including as many relevant datasets as possible. What constitutes a good value for T_S therefore depends on how well the ranking works, which is affected by T_Q and T_C .

It is also good that the confusion regarding threshold values has been cleared up, so that they now are implemented, can be varied and have a consistent description. The T_Q threshold also has a varying effect on how well the system performs, so the related changes to the query processing and the introduction of T_Q has given an improvement of the system. For comparison, setting T_Q to 0.00 effectively cancels the changes that introduced it, and we can see that higher values do indeed improve the results. T_C is also an effective threshold variable, and the interplay between the three threshold variables

means they must be chosen carefully, since the performance varies from very bad to quite good.

Speaking of performance: DataOntoSearch version 2 performs better than version 1, except for the ranking measures of automatic linking. This is only to be expected since my experiments never considered the automatically tagged associations. The manual linking of version 2 performs many times better than version 1, meaning that the changes have worked and brought with them a system more able to respond to users' demands.

7.4.1 Limitations

It is unfortunate that version 1 does not have functioning thresholds, because we may not know how well it might perform with optimized values. We saw that the thresholds dramatically affect DataOntoSearch's performance, making it go from returning very relevant datasets to returning nothing or returning everything. Version 1 also exhibit this trait, which means that the evaluation results we see could just reflect a poor choice of threshold values hardcoded into version 1, and not actual improvements to the algorithms themselves. I would still argue that the fact that the thresholds were neither implemented nor variable in version 1 is a fault with the original system, which should – and probably is – reflected in its evaluation.

The greater limitation here, is the fact that Google Dataset Search experienced a bug when the tests were ran, which stopped some relevant datasets from being retrieved. I therefore do not wish to conclude one way or another regarding whether DataOntoSearch version 2 beats Google or not. You could still make the same argument as with DataOntoSearch version 1, i.e. that the bug is a fault of Google and that this should be reflected in the evaluation, but I think that it is more interesting to know how the engine would perform today than how it performed a particular day in the past. DataOntoSearch version 1 is the way it has been for a couple months now, but Google Dataset Search is continuously developed and changed. This makes it hard to evaluate Google, but it also means there will be temporary hiccups every now and then. I cannot take the blame for evaluating Google on a “bad day,” but I won't take the opportunity to conclude heavily in DataOntoSearch's favour because of it either.

Another reason why it would be unfair to judge Google too harshly, is the fact that the evaluation itself is based on information which is accessible to DataOntoSearch when using the manual tagging. While comparing DataOntoSearch across versions is fine, since the RDF graphs are the same, it may be unfair for Google to be pit against them. Theoretically, if the engineering team at Google were given the same information, they could have adapted the search so it performed better against our evaluation. A counterargument to this is that we very well might have landed on the exact same relevance assessments, had we gone through the datasets one by one and recorded whether they were relevant or not, though there is no way to know that.

A third factor at play here is the choice of threshold values. DataOntoSearch was tuned to some of the queries, which Google Dataset Search cannot do. By design, there were big steps between the different threshold values, to avoid over-specializing to the queries at hand. That said, it would probably be better to have some smaller steps around the extremes, since there is a very big gap from 1.00 and 0.75. Early testing, for comparison, found that the assignment of $T_S = 0.75, T_Q = 0.95, T_C = 0.00$ was optimal, although

$T_S = 0.75$ was set from the start and the other two thresholds were found based on that, so the system did not do so well with the unranked measures. The distinction between training and testing queries played a role in alleviating the unfairness, and you can see that the best combination – $T_S = 0.25, T_Q = 1.00, T_C = 1.00$ – was not chosen.

7.5 Run-time performance

Since only the performance of DataOntoSearch version 2 has been captured, it is not possible to systematically compare it to with the other systems. Anecdotally, the time usage is significantly higher than what most people are used to from typical web search engines, and must be improved in the future.

The real-world runtime per query reported by `perf stat` and by Table 6.21 differs by 1.101 seconds, a quite significant amount of time. This is caused by the difference in what is measured by the two measuring methods. The former method measures the run-time for the entire program, while the latter only measures time spent in the method responsible for performing the search. Specifically, the run-time of the following procedures are factored into the former but not the latter time measurements:

- Fetching ontology and dataset graphs from the data store.
- Parsing and converting graphs into a Python data structure.
- Fetching pre-calculated matrices (CCSM and DCSM) from the data store.
- Loading and converting pre-calculated matrices into NumPy data structures.
- Printing the results to the console window.
- Parsing the query specification file (only done once for the process).

Most of these processes are only repeated when running multiple queries using the CLI, since the underlying data structures are kept across queries run using the webserver process. The time measurements shown in Table 6.21 are therefore likely to be more representative of how users perceive the system’s performance, though with a longer execution time due to fetching dataset metadata.

7.6 Comparison with some other approaches

The overall approach of DataOntoSearch has not changed much between version 1 and version 2. There are mainly some details that have changed, though they have affected the performance greatly.

In Section 2.1.4 on page 7, a taxonomy of semantic search approaches was presented. It can be interesting to classify DataOntoSearch in terms of those dimensions.

- **Retrieval scope: Linked-data-retrieval techniques.** Though an ontology is used, the taxonomy seems to be using the word “ontology” to describe RDF vocabularies, not hierarchies of concepts. Since DataOntoSearch searches among datasets expressed using DCAT, the linked-data-retrieval techniques best describes the system.

- **Query model: Keyword search.**
- **Results type: Entity-centric or document-centric.** The information about each dataset is collected using the entire index, but if multiple sources refer to the same dataset using different RDF identifiers, they will be treated as separate datasets due to there not being any de-duplication mechanism.
- **Data acquisition: Manual collection.** A dataset is added when the user adds a manual dataset-concept association involving it, or when added to the dataset graph manually.
- **Ranking factor: Query-dependent.** The exact ranking factor used by DataOntoSearch is not listed in the taxonomy, but is the similarity between the Query-Concept Similarity Vector and the dataset's Dataset-Concept Similarity Vector.
- **Datasets: Real-world data.** The evaluation was done with real datasets collected from New York State's Socrata and the other publishers they republish from.
- **User interface: Graphical user interface and API.** With DataOntoSearch version 2, APIs are supported in addition to the existing graphical user interface.

Google Dataset Search, described in Section 3.1.1, is different from DataOntoSearch in a number of ways.

- Google's results are entity-centric, using a de-duplication mechanism to understand when multiple sources describe the same dataset [36], and presents a unified view of that dataset with links to the different sources.
- Google obtains data by crawling the web.
- The ranking factors of Google Dataset Search are not known.
- Google Dataset Search only provides a graphical user interface, no API.

There are also some similarities:

- Both systems focus on linked-data-retrieval techniques.
- Both systems use keyword search, letting users type in text in a free-form text box.

The spatio-temporal search described in Section 3.1.2 is more similar to DataOntoSearch. They, too, provide both a graphical user interface and an extensive API. Their query model, however, is closer to faceted search than keyword search, since the user must specify the time period and geographical area in separate text fields, rather than trying to extract the information from a long user query.

Furthermore, the spatio-temporal search and DataOntoSearch complement each other quite well, since the former system filters datasets based on geographical and temporal information with a plain keyword-based search underneath, while DataOntoSearch retrieves and ranks datasets based on their topic and has no concept of time and space. Future work could very well combine the two approaches, making spatio-temporal filtering available as facets while letting DataOntoSearch handle the underlying text search.

7.7 Future work

There are a couple problems and possible future directions that should be the focus of any new endeavors made with DataOntoSearch:

- **Hybrid search:** For queries like “new york state highway public facilities,” the words “new york state” *may* actually have entries in WordNet, but they are not helpful since the ontology contains no geographical taxonomy. Instead, it would be interesting to combine semantic search with traditional keyword-based search, which could help add some precision to the search and better support elements that are not modelled in the ontology.
- **Show why a result is relevant:** A path could be shown to indicate how the dataset relates to the query, e.g. word in query → WordNet synonym → concept → related concept → the dataset. The challenge of this would be to figure out why a dataset is related to the query, since that depends on its cosine similarity, and also how to get this information from algorithms that essentially work like black boxes.
- **Alternatives to WordNet:** Are there alternatives to WordNet, that can be used? Or can WordNet be extended, so domain specific terminology is better represented?
- **Integrate with spatio-temporal semantic search:** The work of Neumaier and Polleres, discussed in Section 3.1.2 on page 18, covers aspects not covered by DataOntoSearch, namely geography and time.
- **Improve CKAN integration:** The semantic search in CKAN is very slow for searches with many results, due to the search mechanism looking up metadata for all datasets. Paging options should be implemented to alleviate this. The semantic search results page should also be able to show matching concepts for the retrieved datasets, and give search suggestions based on the available concepts. There are also many smaller improvements that can be made to the usability of the CKAN extension, some of which are mentioned in its README file¹.
- **Handle search different for automatic tagging:** Right now, only the manually tagged dataset-concept associations were used while working on improving the search procedure. Since the structure of the manual and automatic associations differ quite a lot, the improvements that worked well for manual tagging may not have worked so well for the automatic one. Some custom changes may need to be applied when using the automatic tagging.
- **Better storage solutions:** MongoDB is used as a simple document store, simply because that was what Hagelien used for version 1. There exists custom datastores for RDF, though, which could give the application a boost in processing time and memory usage (since all RDF graphs are currently loaded into memory from MongoDB). This could also solve the current problem with there not being any protection against concurrent changes to the graphs, which can lead to changes being lost because the graph that is saved last overrides any changes anyone else did in the meantime.

¹<https://github.com/tobinus/ckanext-dataontosearch#future-work>

- **Handle private datasets in CKAN:** Private datasets are still made available in DataOntoSearch by the CKAN plugin. Even though they are not shown when searching inside CKAN, DataOntoSearch itself has no concept of datasets being private and will show them all to you when you use it directly.
- **Authentication and authorization in DataOntoSearch:** Right now, all the API endpoints are unprotected. It has not been prioritized since the system has not been put into production, and because access control can be implemented in the web server, e.g. Nginx or Apache. It would still be nice to add a username/password combination to each Configuration, so that there is built-in protection against people screwing around with other CKAN instances' manual tagging.
- **Better use of NumPy functionality:** The matrix calculations currently take a lot of time, most of which is spent working with NumPy matrices. It is possible that great performance improvements can be achieved by using built-in NumPy functionality instead of spending all the time in our own custom Python code. This especially applies to the process of binding concepts to the query, which takes up the majority of time spent processing queries.
- **Handle incremental changes to the index:** Right now, the Concept-Dataset Similarity Matrix is calculated from scratch every time the concept-dataset associations have changed. This is not scalable, especially not when users make one change after another in CKAN. Ideally, the changes made since last time should just be applied to the existing matrix, at least for small changes like a new dataset being added or a new association being made. Changes to the ontology are likely very hard to do incrementally.
- **Simplification and refactoring:** Though I spent a lot of time refactoring the code in the beginning, some parts have not been treated so well afterwards. There may also be instances of overengineering, though I'm not the best person to judge that.
- **Addressing weaknesses mentioned but not addressed:** The section on research motivation, Section 4.1 on page 35, mentions several more weaknesses that I did not have the time to address, like using dataset contents for automatic tagging and improving the ontology with equivalence relationships.
- **Investigate some new methods:** At one point, I got a tips from Hagelien, who had attended a conference and presented DataOntoSearch. Some topics that could be relevant, based on conversations he had had, are B-cube and Entity Alignment F-measure, and also clustering for evaluating parts of the system.

Conclusion

At the start of this project, I was set to explore an ontology-based semantic search system for open data, a system which consists of many words I did not really understand the meaning of back then. Through reading books and literature on related topics, and doing some detective work to figure out what DataOntoSearch is doing and how it is trying to do that, I have gained a greater understanding of the challenges of making open data discoverable and accessible to developers and other users not familiar with the domain in question.

I did not foresee spending so much time refactoring and making the DataOntoSearch runnable, but the experimental approach of my predecessor left me with little choice. Initially thought of as a way to get to know the system while making it easier to improve it later, the problems ran deep and I was still struggling with making sense of it all right until the last few months, with the confusing situation surrounding the threshold variables.

I also did not think so much time would go into researching how to do evaluations, but they are an important part of this project. The usability test in particular took a lot of time to prepare, but from it, we learnt more about the state of DataOntoSearch, and brought in a down-to-earth perspective often missing from semantic search research. It revealed that DataOntoSearch version 1 was not in a good position, and that I had my work cut out for me when finally sitting down to improve it. Specifically, users noted that the system did not look styled, and were unsatisfied with the irrelevant datasets retrieved. This answers our first research question of what users think of DataOntoSearch version 1.

Moving on to the second research question, my first order of action was to make DataOntoSearch itself more easily available. We cannot expect users to seek out a third-party website to perform a search or add dataset-concept links, when they already sit in CKAN and are doing just fine. By making DataOntoSearch's functionality available for use within CKAN, the benefits it brings stand a better chance of actually making a change for users searching after datasets. The work on the CKAN extension has also paved the way for integrating DataOntoSearch with other systems, since any technology can make use of DataOntoSearch's APIs.

Of course, having DataOntoSearch in CKAN is not going to do users any good if its

search performs much worse than alternative search options. Setting a lower threshold for what concepts are associated with the query through WordNet reduces the importance of WordNet's structure. Enriching the associated concepts with related concepts helps with making the query vector more similar to the datasets' vectors, and increases the importance of the hand-crafted ontology which separates DataOntoSearch from other approaches within semantic open government data search. The ontology is also utilized to a greater extent along with WordNet, by overhauling how the query is matched up with concepts. Finally, multi-word concepts can be associated with the user's query, and the knowledge embedded in WordNet is used more now that its multi-word entries are used to calculate the query's similarity to concepts.

The system-oriented evaluation methods confirm that a significant improvement has been made over DataOntoSearch version 1 for manually created dataset-concept associations, confirming that the problem with irrelevant datasets has been addressed – at least to some degree. Though the results indicate that the system surpasses Google Dataset Search as well, unfortunate circumstances sow doubt about how representable Google's performance was for their engine. It is also an open challenge to find a form of systematic evaluation which does not give any search engine an unfair advantage, without spending a huge amount of effort assessing each dataset's relevancy to each query.

All in all, the ontology-based semantic search approach seems to hold some merit, especially now that concepts can be matched with with greater predictability. By further improving DataOntoSearch's capabilities and fitness for use, the system may facilitate interdisciplinarity by granting developers who are well versed in the domain of computer science the ability to find and create new, innovative applications using open data from the transport domain. Though keyword-based approaches like Google Dataset Search are promising, they may not help as much to achieve this vision due to the barrier of entry for users not familiar with the domain-specific language.

Bibliography

- [1] Rashid Ali and M.M. Sufyan Beg. “An overview of Web search evaluation methods”. English. In: *Computers & Electrical Engineering* 37.6 (2011), pp. 835–848. ISSN: 0045-7906.
- [2] Dean Allemang. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. eng. 2nd ed. Amsterdam: Elsevier, 2011. ISBN: 9780123859655.
- [3] Grigoris Antoniou et al. *A Semantic Web Primer*. 3rd ed. London, England: The MIT Press, 2012. ISBN: 978-0-262-01828-9.
- [4] Apache. *Apache Solr - Features*. 2019. URL: <https://lucene.apache.org/solr/features.html> (visited on 05/23/2019).
- [5] Tim Bernes-Lee. *Linked Data - Design Issues*. 2010. URL: <https://www.w3.org/DesignIssues/LinkedData.html> (visited on 05/21/2019).
- [6] Roi Blanco et al. “Repeatable and reliable semantic search evaluation”. eng. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 21.C (2013), pp. 14–29. ISSN: 1570-8268.
- [7] John Brooke. *SUS - A quick and dirty usability scale*. eng. Nov. 1995.
- [8] M. Buranarach et al. “Open data search framework based on semi-structured query patterns”. In: *CEUR Workshop Proceedings 2000* (2017), pp. 13–19. ISSN: 16130073.
- [9] Anila Sahar Butt, Armin Haller, and Lexing Xie. “A Taxonomy of Semantic Web Data Retrieval Techniques”. In: *Proceedings of the 8th International Conference on Knowledge Capture*. K-CAP 2015. Palisades, NY, USA: ACM, 2015, 9:1–9:9. ISBN: 978-1-4503-3849-3. DOI: 10.1145/2815833.2815846. URL: <http://doi.acm.org/10.1145/2815833.2815846>.
- [10] Davide Castellevecchi. “Google unveils search engine for open data”. eng. In: *Nature* 561.7722 (Sept. 5, 2018), pp. 161–162. ISSN: 00280836. DOI: 10.1038/d41586-018-06201-x. URL: <http://search.proquest.com/docview/2116833645/> (visited on 05/26/2019).

-
- [11] Gong Cheng and Yuzhong Qu. “Searching linked objects with falcons: Approach, implementation and evaluation”. In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5.3 (2009), pp. 49–70.
- [12] ckan. *Search and Discovery*. URL: <https://ckan.org/portfolio/search-and-discovery/> (visited on 05/23/2019).
- [13] Mathieu d’Aquin et al. “Watson: A gateway for the semantic web”. In: (2007).
- [14] Stefan Decker et al. “Ontobroker: Ontology based access to distributed and semi-structured information”. In: *Database Semantics*. Springer, 1999, pp. 351–369.
- [15] Li Ding et al. “Swoogle: a search and metadata engine for the semantic web”. In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 2004, pp. 652–659.
- [16] Khadija M. Elbedweihy et al. “An overview of semantic search evaluation initiatives”. eng. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 30.C (2015), pp. 82–105. ISSN: 1570-8268.
- [17] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: <https://www.w3.org/Protocols/rfc2616/rfc2616.txt>.
- [18] Roy T. Fielding and Richard N. Taylor. “Principled Design of the Modern Web Architecture”. In: *ACM Trans. Internet Technol.* 2.2 (May 2002), pp. 115–150. ISSN: 1533-5399. DOI: 10.1145/514183.514185. URL: <http://doi.acm.org/10.1145/514183.514185>.
- [19] Open Knowledge Foundation. *Open Definition 2.1*. URL: <http://opendefinition.org/od/2.1/en/> (visited on 05/21/2019).
- [20] Thomas Fjæstad Hagelien. “A Framework for Ontology Based Semantic Search”. eng. MA thesis. Trondheim, July 2018. URL: <http://hdl.handle.net/11250/2567220>.
- [21] Kotaro Hara et al. “A Data-Driven Analysis of Workers’ Earnings on Amazon Mechanical Turk”. In: (2017).
- [22] Aileen Hay et al. *Federated catalogue service for open transport data*. Trondheim, Nov. 2016.
- [23] Jeff Heflin, James A Hendler, and Sean Luke. “SHOE: A Blueprint for the Semantic Web.” In: *Spinning the Semantic Web* 1 (2003), pp. 1–19.
- [24] Aidan Hogan et al. “Searching and browsing Linked Data with SWSE: The Semantic Web Search Engine”. In: *Journal of Web Semantics* 9.4 (2011). JWS special issue on Semantic Search, pp. 365–401. ISSN: 1570-8268. DOI: <https://doi.org/10.1016/j.websem.2011.06.004>. URL: <http://www.sciencedirect.com/science/article/pii/S1570826811000473>.
- [25] Shanshan Jiang et al. “Ontology-Based Semantic Search for Open Government Data”. In: *13th IEEE International Conference on Semantic Computing, ICSC 2019, Newport Beach, CA, USA, January 30 - February 1, 2019*. IEEE, 2019, pp. 7–15. ISBN: 978-1-5386-6783-5. DOI: 10.1109/ICOSC.2019.8665522. URL: <https://doi.org/10.1109/ICOSC.2019.8665522>.
-

-
- [26] Esther Kaufmann and Abraham Bernstein. "Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases". eng. In: *Web Semantics: Science, Services and Agents on the World Wide Web 8.4* (2010), pp. 377–393. ISSN: 1570-8268.
- [27] Rob Kitchin. *The Data Revolution: Big Data, Open Data, Data Infrastructures & Their Consequences*. eng. London: SAGE Publications Ltd, 2014. DOI: 10.4135/9781473909472.
- [28] Hoa Loranger. *Redesigning Your Website? Don't Ditch Your Old Design So Soon*. eng. Dec. 7, 2014. URL: <https://www.nngroup.com/articles/redesign-competitive-testing/> (visited on 10/10/2018).
- [29] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. eng. Cambridge, 2008. URL: <https://nlp.stanford.edu/IR-book/>.
- [30] Natasha Mathur. *Red Hat drops MongoDB over concerns related to its Server Side Public License (SSPL)*. eng. Jan. 17, 2019. URL: <https://hub.packtpub.com/red-hat-drops-mongodb-over-concerns-related-to-its-server-side-public-license-sspl/> (visited on 06/05/2019).
- [31] Jorge Morales and Andrés Melgar. "Research on Proposals and Trends in the Architectures of Semantic Search Engines: A Systematic Literature Review". In: *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*. 2017 Federated Conference on Computer Science and Information Systems 11 (2017), pp. 271–280.
- [32] Sebastian Neumaier and Axel Polleres. "Enabling Spatio-Temporal Search in Open Data". In: *Journal of Web Semantics 55* (2019), pp. 21–36. ISSN: 1570-8268. DOI: <https://doi.org/10.1016/j.websem.2018.12.007>. URL: <http://www.sciencedirect.com/science/article/pii/S1570826818300696>.
- [33] Jakob Nielsen. *Usability 101: Introduction to Usability*. eng. Jan. 4, 2012. URL: <https://www.nngroup.com/articles/usability-101-introduction-to-usability/> (visited on 10/08/2018).
- [34] Jakob Nielsen. *Usability Engineering*. eng. Academic Press, 1993. ISBN: 0-12-518405-0.
- [35] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. eng. Mar. 19, 2000. URL: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> (visited on 06/05/2019).
- [36] Natasha Noy. *Making it easier to discover datasets*. Sept. 5, 2018. URL: <https://www.blog.google/products/search/making-it-easier-discover-datasets/> (visited on 09/20/2018).
- [37] Natasha Noy and Dan Brickley. *Facilitating the discovery of public datasets*. Jan. 24, 2017. URL: <https://ai.googleblog.com/2017/01/facilitating-discovery-of-public.html> (visited on 09/20/2018).

-
- [38] Sara Ashley O'Brien. *Why Uber and Lyft drivers are striking*. May 8, 2019. URL: <https://edition.cnn.com/2019/05/07/tech/uber-driver-strike-ipo/index.html> (visited on 05/10/2019).
- [39] Eyal Oren et al. "Sindice. com: a document-oriented lookup index for open linked data". In: *International Journal of Metadata, Semantics and Ontologies* 3.1 (2008), pp. 37–52.
- [40] Princeton University. *About Wordnet*. eng. 2010. URL: <https://wordnet.princeton.edu/> (visited on 02/11/2019).
- [41] Amy Schade. *Competitive Usability Evaluations: Learning from Your Competition*. eng. Dec. 15, 2013. URL: <https://www.nngroup.com/articles/competitive-usability-evaluations/> (visited on 10/10/2018).
- [42] Oscar Schwartz. *Untold History of AI: How Amazon's Mechanical Turkers Got Squeezed Inside the Machine*. Apr. 22, 2019. URL: <https://spectrum.ieee.org/tech-talk/tech-history/dawn-of-electronics/untold-history-of-ai-mechanical-turk-revisited-tkktk> (visited on 05/10/2019).
- [43] Alana Semuels. *The Internet Is Enabling a New Kind of Poorly Paid Hell*. Jan. 23, 2018. URL: <https://www.theatlantic.com/business/archive/2018/01/amazon-mechanical-turk/551192/> (visited on 05/10/2019).
- [44] Maximilian Speicher. *What is Usability?* eng. 2015. URL: <http://www.qucosa.de/fileadmin/data/qucosa/documents/15994/CSR-2015-02.pdf>.
- [45] Xiaolong Tang et al. "Ontology-Based Semantic Search for Large-Scale RDF Data". In: *Web-Age Information Management*. Ed. by Jianyong Wang et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 570–582. ISBN: 978-3-642-38562-9. DOI: 10.1007/978-3-642-38562-9_58.
- [46] Giovanni Tummarello et al. "Sig. ma: Live views on the web of data". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 8.4 (2010), pp. 355–364.
- [47] W3C Working Group. *SKOS Simple Knowledge Organization System Primer*. eng. Aug. 18, 2009. URL: <https://www.w3.org/TR/skos-primer/> (visited on 05/21/2019).
- [48] W3C World Wide Web Consortium. *Data Catalog Vocabulary (DCAT)*. eng. Jan. 16, 2014. URL: <https://www.w3.org/TR/vocab-dcat/> (visited on 05/21/2019).
- [49] W3C World Wide Web Consortium. *RDF 1.1 Concepts and Abstract Syntax*. eng. Feb. 25, 2014. URL: <https://www.w3.org/TR/rdf11-concepts/> (visited on 05/21/2019).
- [50] Zhibiao Wu and Martha Palmer. "Verbs semantics and lexical selection". In: *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 1994, pp. 133–138.
- [51] Haiping Xu and Arturo Li. "Two-Level Smart Search Engine Using Ontology-Based Semantic Reasoning." In: *SEKE*. 2014, pp. 648–652.
-

Appendices

Appendix A

Material provided to test users in pre-study

Test users should be provided with a test description and a manual, so they know how to use the different systems. They should also be provided with the task descriptions, one at a time. This section provides a copy of this material.

The test description is a short introduction to the test. Next, the task descriptions are given. An internal note about the tasks and their possible solutions is also included.

Finally, the manual provided to test users as an introduction to the systems is given.

A.1 Test description

This is a test of two systems' usability, aiming to learn how easy they are to use, and what problems they have. This project is a pre-study, and is a part of a master thesis. The master thesis is about a search engine meant to help software developers find datasets. Datasets can for example be "Bus schedule in Trondheim" or "CO₂ levels in Tromsø."

As a part of the test, you are asked to:

1. Try to finish three tasks with one search engine, and think aloud when doing so.
2. Answer a short questionnaire about the search engine you just tried out.
3. Do step 1-2 using another search engine.
4. Answer a questionnaire about which search engine you enjoyed the most.

It is important for us to emphasise that *you* are not being tested. What we are testing, is the *system*. Any problems you encounter during the test, will be considered problems with the system, not you, so please relax!

Information that will be collected

The following information will be collected:

- Written notes by researchers, based on what you say
- Answers to all three questionnaires
- Task solutions found
- Time spent on each task
- Video of the computer screen (without sound)
- What study program you attend

No personal information will be collected. As an example, name and age is not collected, and no voice recording is made. Due to this, the regulations related to personal data do not apply.

Participation is voluntary

Participation in the project is voluntary. **If you choose to participate, you can withdraw your consent at any time without giving a reason.** There will be no negative consequences for you if you choose not to participate or later decide to withdraw. Specifically, I will not hold it against you if you do so. **Your participation or lack thereof will be kept confidential and not be shared with others.** You, on the other hand, are free to speak to others about this.

A.2 Task descriptions

Task 55

You are a software developer living in the city of New York. You would like to create a computer program telling you the nearest place where you can go to ride the underground railway. You decide to use a dataset so your computer program can retrieve information automatically.

Can you find a dataset you could use to solve this problem?

Task 47

You are a software developer working for a tourist agency focusing on tourists who would like to experience the *state* of New York, but not the famous New York City itself. To help them get around, the agency would like to display information about public facilities along the highways at which travelers can stop and take a break from driving (“rasteplass”) – specifically the ones which the state of New York have responsibility for. You decide to use a dataset so your website can retrieve information automatically.

Can you find a dataset you could use to solve this problem?

Task 68

You are a software developer working for a travel agency focusing on travels inside all of the United States of America. Customers visit your website, and write where they would like to travel from, where they would like to travel to, and a range of dates where they would like to travel. Right now you only show available flights or options for renting a car, but the travel agency wants to include possible train departures as well. You decide to use a dataset so the website can find available train departures automatically.

Can you find a dataset you could use to solve this problem?

A.2.1 Internal notes about the tasks

Summary of the tasks:

1. Underground Railway
2. Rest Areas
3. Train schedule

The following are example solutions to the tasks. They are not exclusive; whether a dataset is correct or not need to be assessed to account for potential confusing formulations of the task descriptions.

1. “NYC Transit Subway Entrance And Exit Data”
2. “Rest Areas Across NY”
3. Though there are datasets for tracks and stations, Amtrak does not seem to have a dataset for train schedules.

A.3 Manual

In this pre-study, we focus on search engines meant to help software developers and researchers find datasets. Datasets can for example be “Bus schedule in Trondheim” or “CO₂ levels in Tromsø,” and can be fed into computer programs or be analyzed in spreadsheets.

Datasets can be published by anyone, but we limit ourself to datasets published by “official” entities. For example a municipality (“kommune”), or the government. Currently, datasets are published by different entities on different platforms. They are then collected and presented on federated platforms, which present datasets from multiple sources in a single place.

A.3.1 Dataset publishers

The following dataset publishers are relevant to the tasks in this test:

- **data.cityofnewyork.us**: Open data from the City of New York (NYC).
- **data.buffalony.gov**: Open data from the city of Buffalo, in the state of New York.
- **data.ny.gov**: Open data from the state of New York itself. In addition, datasets from the two publishers above are included.
- **data.gov**: Open data from the entirety of the United States of America. This is a federated catalog, including all three publishers above.

A.3.2 Search engines

There are two search engines used in this test.

- **DataOntoSearch**: Semantic search using concepts organized in a hierarchy.
- **Google Dataset Search**: Keyword search focused on datasets.

They are described in more detail below.

DataOntoSearch

This search engine uses an ontology. An ontology is an overview over concepts in a field, in our case the field of transport. There, concepts are organized in a hierarchy, so that one concept is at the top, with more specialized concepts below it, and even more specific specializations below them again.

For example, “Entity” can be a top concept, with “Abstraction” as one specialization, “Communication” as a specialization of that again, “Geographical information” as a specialization of “Communication,” and finally “Map” as a specialization of “Geographical information.” See Figure A.1 for a visual illustration of this hierarchy.

When you search, your query is compared to the concepts to find which concepts are most relevant. All datasets have also been through this process, and the relevant datasets

-
- Entity
 - Abstraction
 - APIDescription
 - Agreement
 - Communication
 - GeographicalInformation
 - Map
 - Prognosis
 - Forecast

Figure A.1: An excerpt from the ontology hierarchy.

are found by looking at the datasets that are relevant to the same concepts as your query. Thus, concepts act as the link between your query and datasets.

Note that this search is restricted to datasets from the data.ny.gov publisher.

A checkbox is found on the search page. Checking this will show you what concepts your query matched with, and also what concepts each dataset is connected to.

Google Dataset Search

Please refer to the “About” link on the Google Dataset Search page.

Note that Google supports a `site:` operator, which lets you restrict your search to one of the dataset publishers mentioned earlier.

Appendix B

Example of code quality improvements

The two code listings below demonstrate how the method responsible for performing the search was changed during the refactoring work described in Section 5.2.1.

Search procedure from Hagelien

```
def search_query(self, query, cds_name="all"):
    sv = self.get_scorevec(query)
    significant = sorted(list(zip(list(sv.columns), sv.as_
        ↪ matrix()[0])), key=lambda x : x[1],
        ↪ reverse=True)[:5])
    df = sv.append(self.cds[cds_name])
    data = cosine_similarity(df)
    df1 = pd.DataFrame(data, columns=df.index,
        ↪ index=df.index)
    f = df1.loc[query].sort_values(ascending=False)[1:]
    relvec = []
    for x in f.index:
        data = (list(zip(self.cds[cds_name].loc[x].index,
            ↪ self.cds[cds_name].loc[x].as_matrix())))
        data = sorted(data, key=lambda x:x[1],
            ↪ reverse=True)[:5]
        relvec.append(data)
    xs = zip(f.tolist(), map(self.get_dataset_info,
        ↪ list(f.index)), relvec)
    return ([x for x in xs if x[0] > 0.75], significant)
```

Improved search procedure (not the most recent version)

```
def search_query(
```

```

        self,
        query,
        cds_name="all",
        qc_sim_threshold=0.0,
        score_threshold=0.75
    ):
        """
        Perform a search query.
        Args:
            query: Search query to use.
            cds_name: Name of concept-dataset tagging to use
            when retrieving datasets.
            qc_sim_threshold: Lower threshold for how similar a
            word in the query must be to a concept label in order
            for that concept to be considered relevant to the
            query.
            score_threshold: Lower threshold for how similar a
            dataset must be to the query to be included in the
            result. This effectively decides how many datasets are
            included.
        Returns: A tuple. The first item is a list of
            SearchResult that matched, sorted with the most similar
            results first. The second item is a list of the top
            five concepts that were matched with the query.
        """
        # Calculate the query's similarity to our concepts
        query_concept_sim =
            self.calculate_query_sim_to_concepts(
                query,
                qc_sim_threshold
            )

        # What were the most similar concepts?
        most_similar_concepts = self.sort_concept_similarities(
            self.get_concept_similarities_for_query(
                query_concept_sim
            )
        )[:5]

        # How similar are the datasets' similarity to the
        query's similarity?
        dataset_query_sim = self.calculate_dataset_query_sim(
            query_concept_sim,
            cds_name,
            query,

```

```
)

# Put together information for the search results page
results = list()
for dataset, similarity in dataset_query_sim.items():
    # Only consider the most relevant datasets
    if similarity < float(score_threshold):
        continue
    results.append(SearchResult(
        score=similarity,
        info=self.get_dataset_info(dataset),
        concepts=self.get_most_similar_concepts_for_
        ↪ dataset(
            cds_name,
            dataset
        ),
    ))
return results, most_similar_concepts
```

Documentation

This appendix includes the technical documentation that was written for the CKAN extension and for DataOntoSearch. The README file for DataOntoSearch was started on during the refactoring work, but has been extended during the improvement work. The rest was written along with the systems they describe.

The documentation here was originally written using Markdown or restructuredText format. The original versions can be viewed at the following locations:

1. <https://github.com/tobinus/OTD-semantic-framework/blob/master/README.md>
2. https://github.com/tobinus/OTD-semantic-framework/blob/master/dataset_tagger/app/API%20Documentation.md
3. <https://github.com/tobinus/OTD-semantic-framework/blob/master/ontosearch/app/API%20Documentation.md>
4. <https://github.com/tobinus/ckanext-dataontosearch/blob/master/README.rst>

Note that the documentation for the CKAN extension is based on a template given by CKAN.

C.1 README of DataOntoSearch

Ontology Based Semantic Search for Open Transport Data

This repository contains source-code and stubs of code used in the development of the prototype **DataOntoSearch**. Its purpose is to help developers find the datasets they want to use, by not requiring them to use the exact same words in their search queries as those used by the dataset publishers.

C.1.1 Preparing

This guide assumes you use Pipenv, but you may also use virtualenv/pip directly using the requirements.txt files (though support for this may be dropped at a future point).

Reading this guide: Any Python commands in the instructions assume you already have the virtual environment loaded. With *Pipenv*, you do this by running `pipenv shell`, or you can prepend `pipenv run` to whatever you want to run inside the virtual environment. With Pip and Virtualenv, you do this by sourcing the `activate` inside the virtual environment folder. For example, if the virtual environment folder is called `venv`, you run `. venv/bin/ activate .`

1. (Fork and) clone this repository, so you have a copy locally
2. Install Pipenv (if so desired)
3. While in this directory, run `pipenv install .` Pip+Virtualenv users should set up the virtual environment and install from `requirements.txt` now
4. Follow the link in `ordvev/README.md` and extract the files into the `ordvev/` directory
5. Configure (and install if you haven't) MongoDB so you have a user (with password) which can access it. You will also need to enable authentication. See for example the official guide from MongoDB.
6. Create a file called `.env` in this directory, where you define the variables:
 - `DB_USERNAME`: Username to use when logging in to MongoDB
 - `DB_PASSWD`: Password to use when logging in to MongoDB
 - `DB_HOST`: Name of server where MongoDB runs. Defaults to `localhost`
 - `DB_NAME`: Name of database to use in MongoDB. Defaults to `ontodb`

To define for example `DB_USERNAME` to be `john`, you would write:

```
DB_USERNAME=john
```

These can also be set as environment variables.

7. Download the different data for `nlTK`, used for word tokenizing, removing stop words and the like. You can do this by running `python dataontosearch.py nltk_data .`

C.1.2 Usage

There is only one script you need to care about, namely `dataontosearch.py`. It has many subcommands that can be invoked, much like Django's `manage.py` command or the `git` utility.

Documentation for the available subcommands and their arguments is not presented here, instead you should use the `--help` flag to access the built-in help. For example, to see the available subcommands, run:

```
python dataontosearch.py --help
```

Nothing is actually done when you use the `--help` flag, it simply prints the help information and exits. Absolutely all subcommands accept the `--help` flag, so please use it to your heart's content!

Below, some general information is provided about how to use DataOntoSearch. Afterwards, the process of getting the search system into a usable state is described.

Entities

There are four types of RDF graphs in the application's database:

1. **Ontology:** Concepts and their relation to one another
2. **Dataset:** Available datasets in DCAT format
3. **Similarity:** Similarity graph, linking datasets to concepts manually
4. **Autotag:** Similarity graph, linking datasets to concepts automatically

These entities are managed through their own subcommands, in a fairly consistent way. They all have a canonical version, which is used by default. It is also possible to load a custom graph into the database.

In addition, there are different kinds of matrices used:

1. **Concept-concept similarity matrix:** Similarities between concepts, calculated using Wu-Palmer similarity.
2. **Concept-dataset similarity matrix for manual tagging:** Similarities between datasets and concepts, enriched by the concept-concept similarity matrix.
3. **Concept-dataset similarity matrix for automatic tagging:** Same as above, except this uses the *autotag* graph rather than the *similarity* one.

Since they are all derivatives of graphs, they are automatically created when needed, and re-created whenever the graph they directly depend on (ontology, similarity and auto-tag respectively) changes or is otherwise updated.

Choosing graphs to use

For each of the graphs mentioned above, the application's choice of graph to use is done like this:

1. Has a UUID been specified using CLI options or the like (where available)? If so, use the specified graph.
2. Has a UUID been specified using the environment variable named `<GRAPH-NAME>_UUID`, where `<GRAPH-NAME>` is the upper-cased name of the graph? For example, `ONTOLOGY_UUID` or `DATASET_UUID`. If so, use the specified graph.

-
3. If no graph has been specified this far, then whatever graph is returned first by MongoDB is used. The system will warn you about this, since there is no guarantee that the same graph would be returned at a later time.

Defining environment variables in `.env` As a shortcut for defining environment variables (step 2 above), the DataOntoSearch supports the use of a `.env` file (sometimes called "dotenv"). There you can define environment variables which could potentially be tiresome to define in your shell every time. The system will print a message whenever it reads from a `.env` file; if you don't receive such a message then you can assume it wasn't read.

Caveat: Pipenv will read the `.env` file when you create a shell (`pipenv shell`). Changes you make to the `.env` file will not be picked up before you exit and re-enter the Pipenv shell. Even though DataOntoSearch reads from `.env` itself, it will not override environment variables already set by your shell, so the potentially outdated values set by Pipenv will override those read from `.env` at runtime.

The Configuration entity There is one exception to the procedure above, namely the **Configuration** mechanism. A Configuration is simply a selection of graphs, one for each type, which is stored alongside a label. The purpose of a Configuration is to allow clients to connect to DataOntoSearch and simply specify their configuration, from which the system knows what graphs to load. This way, different organizations may connect to the same DataOntoSearch process, yet use different ontologies, datasets and taggings.

Configurations are used by the dataset tagger (`dataset.tagger`) and the search itself (`serve`) along with related commands (`search`, `multisearch` and `evaluate`). The selection of a Configuration follows the exact same procedure as for graph selection above, using the `CONFIGURATION_UUID` environment variable when the user does not specify a Configuration themselves, or using whatever Configuration MongoDB returns first.

The Configuration is not used for any other commands. The reason is that a **Configuration requires there to be one graph of each type**, since it will be pointing to one graph of each type. Therefore, you cannot create a Configuration before everything else has been set up; it will generally be the last thing you do before the search is ready.

Suggested approaches

- Keep only one version of each graph type, so it is obvious which one is picked by MongoDB.
 - You don't need to specify what graph to use this way.
 - However, if there are more than one graph, you may encounter subtle bugs due to an unexpected graph being used.
- Specify the UUID of the graph to use in the `.env` file, using the appropriate environment variables.
 - Works well when you'd like to switch between different graphs.
 - You may override the `.env` variables on the command line.

-
- Though this means you must manage the UUIDs of graphs.
 - You can use multiple files and change between them by renaming one of them to `.env`, and let the others have other names when not in use.

Setup

Before the search engine can get up and running, there are a couple processes that must be run. Specifically:

1. Pre-process ontology
 1. Upload ontology
2. Pre-process datasets
 1. Import (new) datasets
 2. Perform manual linking to concepts
 3. Perform automatic linking to concepts
3. Run search

Below, you'll find instructions for running these processes.

Pre-processing of ontology

Upload ontology

1. Run `python dataontosearch .py ontology create`

There are other commands you can use to manipulate and display ontologies, run `python dataontosearch .py ontology --help` for a full list.

Pre-processing of datasets

Import (new) datasets You may decide to import a set of dataset, or all datasets available to a CKAN instance or another compatible system. You will typically do this if you want to use the automatic tagger or the spreadsheet approach below:

1. Run `python dataontosearch .py dataset create --ckan <CKAN-URL>` where you replace `<CKAN-URL>` with the base URL to your CKAN instance. The instance must also expose RDF information about each dataset, using the `ckanext-dcat` plugin.

If you use another system which exposes the datasets using the DCAT vocabulary, you can import its data by downloading the RDF and using the `--read` flag with `python ↪ dataontosearch .py dataset create`.

Alternatively, you may start with an empty graph. This is useful when you intend to add datasets gradually through the `dataset.tagger`, which automatically downloads dataset metadata for unknown datasets. Simply use the `--empty` flag.

As always, run `python dataontosearch .py dataset create --help` to see your options.

Perform manual linking to concepts You may use the existing tags for the Open Transport Data project, if you use the very same datasets. Simply run:

```
python dataontosearch .py similarity create
```

Alternatively, you can do custom tagging. There are two ways:

- **Using a spreadsheet:** This approach is preferred for mass-tagging of many datasets at once.
 1. Ensure the newly created ontology and dataset graphs are set to be used. See the instructions above on "Choosing graphs to use".
 2. Use `python dataontosearch .py similarity csv_prepare` to generate a CSV file which can be filled in.
 3. Read the instructions found when running `python dataontosearch .py similarity ↔ csv_parse --help` for information on the two options you have for filling in the CSV file.
 4. Fill the CSV file with your taggings, potentially using `python dataontosearch ↔ .py ontology show_hier` to see a list of available concepts.
 5. Export a new CSV file with your manual tagging.
 6. Use `python dataontosearch similarity csv_parse` to make the CSV file into an RDF graph, which you should save to a file.
 7. Use the `--read` option with `python dataontosearch similarity create` to store the newly create graph in the database.

- **Using online application:** This approach is preferred when incrementally adding datasets, for example when used with a running CKAN instance.
 1. Run `python dataontosearch .py similarity create --empty` to create a new, empty similarity graph.
 2. You must set up a Configuration before running the dataset tagger. This requires you to have an autotag graph already, so you may choose to continue with this set-up procedure. Alternatively, you can create a new, empty autotag graph by running `python dataontosearch .py autotag create --empty`. Then you can create a new Configuration using this empty autotag graph.
 3. Run `python dataontosearch .py dataset_tagger`
 4. Now, you can use the included interface for tagging datasets, or use the API with e.g. the CKAN DataOntoSearch plugin to tag datasets. Follow these steps to use the built-in interface:
 5. Visit `http://localhost:8000` in your web browser.
 6. Fill in the UUID of the Configuration you have created (in step 2).
 7. Follow the instructions to tag datasets.

Perform automatic linking to concepts Run the following:

```
python dataontosearch .py autotag create
```

Again, append `--help` to see available options, for example options for using English WordNet instead of Norwegian (the default).

WARNING 1: The autotag script requires around 2 GB of RAM(!) for the Python process when using the Norwegian OrdNet, meaning that it may get killed on systems with not enough memory. You may choose to run the autotag process on a different system, instead of on the server intended for serving the search (for example, use `generate` instead of `create`, then transfer the generated file).

WARNING 2: This script can take a long time to run, like 45 minutes or even an hour. It is not able to continue where it left out, so you might want to run this in `tmux`, `screen` or something similar when running on a server over SSH (so the process survives if your SSH connection ends).

Create Configuration

Before you can search, you must create a Configuration to be used by the search engine. Ensure that the correct graphs will be chosen by running:

```
python dataontosearch .py configuration create <LABEL NAME> --preview
```

Replace `<LABEL NAME>` with a name for this Configuration so you can understand which Configuration is used for what purpose later on.

This prints the UUID of the graphs that will be chosen for the new configuration. Use the command line arguments available or set the environment variables if the wrong graphs are chosen, or there is a mismatch between the graphs (the similarity and autotag graphs must have been made using the same dataset and ontology graphs).

When satisfied, remove the `--preview` flag to actually create the Configuration.

Run search

Now that you have a Configuration to use, you can start the search process:

1. Run `python dataontosearch .py serve`
2. The webserver will perform some indexing, creating necessary matrices. It will give you a signal when it's done
3. You may now search by visiting `http://localhost:8000`

Alternatively, you may search using the command line interface directly. Run `python dataontosearch .py search --help` for more information.

Note: The web interface only allows access to the default Configuration. The API, however, allows the user to specify a Configuration to use. With this setup, new Configurations can be added or changed without restarting the web server. You will need to re-generate the matrices, however, since it cannot be done inside a request (the request would be aborted before the calculations are done). You can do this by running `python dataontosearch .py matrix`, which you might want to run periodically so changes in the manual tagging and dataset graphs are picked up. You can add it as a recurring task in

a crontab, though you'll need to point to the python executable located in your virtualenv, not just the system-wide python.

Do you want to run multiple queries for machine processing, while varying available thresholds and such? Use the `python dataontosearch .py multisearch` subcommand for this. See its `--help` information for *many* details.

An evaluation subcommand is built on top of the `multisearch` command, allowing you to run systematic evaluations. See `python dataontosearch .py evaluate --help` for an introduction.

C.2 API Documentation of dataset tagger

The endpoints available through the `dataset_tagger` are documented below.

URL variables describe parts of the URL which are variable, denoted by angle brackets. Currently, only the Configuration UUID is in use. It must be set so that the application knows what set of graphs to use (see the entity description in the root README).

Query parameters describe what GET query parameters are available to use.

Similarly, **JSON payload** describes the format of the JSON you must send as the POST payload. The names found on <https://www.json.org> are used to describe the types. Note that the Content-Type header must be set appropriately, for example `application/json`.

Finally, **Response JSON** describes the JSON returned by the `dataset_tagger` in response to your query.

Note: Usually, there are two ways of specifying a dataset.

- You can use the RDF IRI, which identifies the dataset in the RDF. This is called `dataset_id`, but should not be confused with the ID associated with the dataset in e.g. CKAN.
- Or you can use a URL at which the system can download RDF information about the dataset. This is called `dataset_url`, but should not be confused with the RDF IRI, URI or URL. The RDF IRI is extracted by finding the first dataset described in the downloaded graph.

”RDF URI” and ”RDF IRI” are used interchangeably, though the latter is more correct (see the RDF spec).

C.2.1 Overview

Method	Endpoint	Purpose
GET	<code>/api/v1/<uuid>/concept</code>	Retrieve concepts available to you
GET	<code>/api/v1/<uuid>/tag</code>	Retrieve existing tags connecting dataset and concepts
POST	<code>/api/v1/<uuid>/tag</code>	Tag a dataset with a concept
DELETE	<code>/api/v1/<uuid>/tag</code>	Remove a tag connecting a concept to a dataset
DELETE	<code>/api/v1/<uuid>/dataset</code>	Remove a dataset and all associated tags

C.2.2 GET `/api/v1/<uuid>/concept`

Retrieve the concepts in the ontology.

URL Variables

Variable	Description
uuid	UUID of the Configuration to use

Query parameters

No parameters are accepted.

Response JSON

Parameter	Type	Description
<i>root</i>	object	Each member of this object represents a concept
<URI>	string	URI is the RDF URI of a concept in the ontology. The value is a human-readable label for this concept

C.2.3 GET /api/v1/<uuid>/tag

Retrieve what concepts the datasets have been tagged with.

URL Variables

Variable	Description
uuid	UUID of the Configuration to use

Query parameters

Parameter	Type	Description
dataset_id or dataset_url	string	Optional. If not provided, tags for all datasets are retrieved. Use <i>dataset_id</i> if you have the RDF IRI of the dataset to retrieve tagged concepts for, or use <i>dataset_url</i> if you have the URL from which DCAT RDF about the dataset can be downloaded

Response JSON

If **dataset_id** or **dataset_url** was provided:

Parameter	Type	Description
<i>root</i>	object	Represents the specified dataset
title	string	Title of the specified dataset
concepts	array of object	List of concepts associated with the specified dataset
concepts[].uri	string	RDF URI of this concept
concepts[].label	string	Human readable label for this concept

If neither **dataset_id** nor **dataset_url** were provided:

Parameter	Type	Description
<i>root</i>	object	Each member of this object represents a dataset
<URI>	object	URI is the RDF URI of a dataset for which a tag exists
<URI>.title	string	Title of this dataset
<URI>.concepts	array of object	List of concepts associated with this dataset
<URI>.concepts[].uri	string	RDF URI of this concept
<URI>.concepts[].label	string	Human readable label for this concept

C.2.4 POST /api/v1/<uuid>/tag

Tag a dataset with a related concept. If the dataset has not been encountered yet, it will be added to the data store.

URL Variables

Variable	Description
uuid	UUID of the Configuration to use

JSON payload

Parameter	Type	Description
<i>root</i>	object	
dataset_url	string	URL at which RDF DCAT information about the dataset can be found. Used to identify which dataset concept shall be associated with, and to download metadata if this dataset has not been seen before. Because of this last usage, it is not possible to specify the <code>dataset_id</code> directly
concept	string	Either RDF URI or the label of the concept to associate with <code>dataset_url</code>

Response JSON

Parameter	Type	Description
<i>root</i>	object	
success	bool	true if the dataset was tagged successfully, false if an error occurred
id	string	ID of the newly added tag. Only present if success is true
message	string	Error message. Only present if success is false

C.2.5 DELETE /api/v1/<uuid>/tag

Disassociate the dataset with the concept, removing any tags connecting the two.

URL Variables

Variable	Description
uuid	UUID of the Configuration to use

JSON payload

Parameter	Type	Description
<i>root</i>	object	
dataset_id or dataset_url	string	The dataset to disassociate with concept. Use <code>dataset_id</code> if you have the RDF IRI of the dataset in question, or use <code>dataset_url</code> if you have the URL from which DCAT RDF about the dataset can be downloaded
concept	string	Either RDF URI or the label of the concept to disassociate with <code>dataset_id</code> / <code>dataset_url</code>

Response JSON

Parameter	Type	Description
<i>root</i>	object	
success	bool	true if no error occurred. Note that it is not checked whether any such tag actually existed; not removing any tags is still considered a success

Note: On error, this endpoint currently raises an exception and fails with a 500 Internal error status message.

C.2.6 DELETE /api/v1/<uuid>/dataset

Remove all tags associated with the specified dataset, and remove it from the data store.

URL Variables

Variable	Description
uuid	UUID of the Configuration to use

JSON payload

Parameter	Type	Description
<i>root</i>	object	
dataset_id or dataset_url	string	The dataset to remove all traces of. Use <code>dataset_id</code> if you have the RDF IRI of the dataset in question, or use <code>dataset_url</code> if you have the URL from which DCAT RDF about the dataset can be downloaded

Response JSON

Parameter	Type	Description
<i>root</i>	object	
success	bool	true if no error occurred. Note that it is not checked whether any such dataset or tags existed; not removing any dataset or tags is still considered a success

Note: On error, this endpoint currently raises an exception and fails with a 500 Internal error status message.

C.3 API Documentation of search webserver

The endpoint available through the DataOntoSearch is documented below.

The documentation format is similar to that of `dataset.tagger`.

C.3.1 Overview

Method	Endpoint	Purpose
GET	<code>/api/v1/search</code>	Perform a search

C.3.2 GET `/api/v1/search`

Perform a semantic search using DataOntoSearch.

Query parameters

Parameter	Type	Description
q	string	Query to perform
c	string	Optional. The Configuration to use. When not provided, the default configuration is used (the same as for the web interface)
a	number	Optional. Set to 1 in order to use the automatically tagged data, instead of the manual tags (the default)
d	number	Optional. Set to 0 to avoid retrieving metadata about the matched datasets. The title and description fields will be null in that case
ic	number	Optional. Set to 0 to avoid retrieving concepts related to the query and datasets
qcs	float	Optional. The query-concept similarity threshold, default: 0.0
qds	float	Optional. The query-dataset similarity threshold, default: 0.75

Response JSON

Parameter	Type	Description
<i>root</i>	object	
concepts	array	The concepts regarded as the most similar to the query, sorted with the most relevant first
concepts []	object	One matching concept
concepts []. uri	string	The RDF IRI of this concept
concepts []. label	string	The preferred label for this concept
concepts []. similarity	number	The similarity score between the query and this concept
results	array	The datasets regarded as the most similar to the query, sorted with the most relevant first
results []	object	One matching dataset
results []. score	number	The similarity score between the query and this dataset
results []. title	string	This dataset's title
results []. description	string	This dataset's description
results []. uri	string	This dataset's RDF IRI
results []. concepts	array	The concepts regarded as the most similar to this dataset (independently of the query)
results []. concepts []	object	One related concept
results []. concepts []. uri	string	The RDF IRI of this concept
results []. concepts []. label	string	The preferred label for this concept
results []. concepts []. similarity	number	The similarity score between this dataset and this concept

C.4 README of ckanext-dataontosearch

Extension for integrating CKAN with DataOntoSearch.

DataOntoSearch is a project which aims to make it easier to find datasets, by using a domain-specific ontology to find similar datasets. The software is run as a separate server, which other projects like CKAN can connect to.

There are two separate plugins provided with this extension. `dataontosearch_tagging` ↪ provides a way of associating datasets with concepts in the ontology. (Each such association is internally called a "tag", which should not be confused with the traditional tags CKAN provide.) `dataontosearch_searching` provides an integrated way of searching using DataOntoSearch.

The extension adds a link you can follow when editing datasets. From there, you can change what concepts are connected to what datasets.

The extension also adds a link to the alternative search method. Following it lets you search using DataOntoSearch.

Important

This extension does not work by itself. It must be paired with a separately deployed version of DataOntoSearch.

Attention

Both this and DataOntoSearch should be considered experimental. The majority of the work is done by master students who are not affiliated with the project after their involvement ends.

C.4.1 Requirements

This plugin was developed for CKAN version 2.8. We have not checked what other versions it works with, but it does use features introduced in version 2.7.

C.4.2 Installation

To install `ckanext-dataontosearch`:

1. Ensure that the `ckanext-dcat` extension is installed.
2. Ensure that CKAN can accept multiple requests in parallel. For example, if you use `gunicorn` to run your application, you could use the `-w` flag to specify more than 1 worker: `gunicorn -w 4` (This is necessary because this extension's request to DataOntoSearch might cause DataOntoSearch to make a request back to CKAN, so the applications would end up waiting for each other in a deadlock.) Note that the debug setting must be set to `false` for CKAN to work in parallel.
3. Activate your CKAN virtual environment, for example:

```
. /usr/lib/ckan/default/bin/activate
```
4. Install the `ckanext-dataontosearch` Python package into your virtual environment:

```
pip install ckanext-dataontosearch
```

5. Add `dataontosearch_tagging` and `dataontosearch_searching` to the `ckan.plugins` setting in your CKAN config file (by default the config file is located at `/etc/ckan/default/production.ini`). Both are not required, any one of them can be used alone, but that is rather uncommon. They need to be listed after the `dcate` plugins.

6. Add required settings:

```
# Base URL where dataset_tagger is running
ckan.dataontosearch.tagger_url = https://example.com/
    ↪ tagger
```

```
# Base URL where the search for DataOntoSearch is
    ↪ running
ckan.dataontosearch.search_url = https://example.com/
    ↪ search
```

```
# The DataOntoSearch Configuration to use
ckan.dataontosearch.configuration = 5
    ↪ c7ea259c556bb42803fa17e
```

7. Restart CKAN. For example if you've deployed CKAN with Apache on Ubuntu:

```
sudo service apache2 reload
```

C.4.3 Config Settings

The required settings are described in the installation guide. In addition to those, you may specify the login used when connecting to DataOntoSearch:

```
# Username and password to use when querying and tagging
    ↪ datasets in
# DataOntoSearch (HTTP Basic Authentication)
# (optional, default: no credentials).
ckanext.dataontosearch.username = aladdin
ckanext.dataontosearch.password = opensesame
```

In addition, you can also tell the extension to use the autotagged similarity graph when searching, instead of the manual tags:

```
# Whether to use the autotagged graph instead of the manual
    ↪ one when
# searching (optional, default: no).
ckan.dataontosearch.use_autotag = yes
```

C.4.4 Development Installation

To install `ckanext-dataontosearch` for development, activate your CKAN virtualenv and do:

```
git clone https://github.com/tobinus/ckanext-dataontosearch
↪ .git
cd ckanext-dataontosearch
python setup.py develop
pip install -r dev-requirements.txt
```

C.4.5 Future Work

There are plenty of things that should be improved. Here are some of them:

- Integrate concept viewing/editing with the dataset type of view, so the tabs don't disappear once you click on "Concepts".
- Some styling improvements can be done to make it look more appealing and be easier to use.
- Give feedback to the user when they save concept changes successfully.
- Use progress indicator of some kind when the user submits concept changes, and stop them from submitting more than once.
- Give the user an idea of how the concepts relate to one another in a hierarchy, instead of just a flat list. They should only use the most relevant, specific concepts, and not try to fit many "similar" concepts, like you would with tags or search words.
- Give the user more context for each concept. There exist alternate labels that sometimes indicate what other areas that concept is covering, and some even have text that explain and show how to apply that concept. This would require changes to the `dataset_tagger` API in `DataOntoSearch` to make the information available to `ckanext-dataontosearch`.
- Separate the two different plugins into two different Python files, per the CKAN recommendations (to avoid problems with files loading out of order).
- Add translations.

There are also some TODO notes in the source code.

C.4.6 Running the Tests

Note

No tests have been written for this project yet.

To run the tests, do:

```
nose tests --nologcapture --with-pylons=test.ini
```

To run the tests and produce a coverage report, first make sure you have coverage installed in your virtualenv (`pip install coverage`) then run:

```
nosetests --nologcapture --with-pylons=test.ini --with-coverage --cover-package=ckanext.dataontosearch --cover-inclusive --cover-erase --cover-tests
```

C.4.7 Releasing a New Version of ckanext-dataontosearch

Note

Publishing on PyPI under the same name (ckanext-dataontosearch) is only possible if you receive rights from one who already has access. You should be able to make contact through an author's GitHub user.

ckanext-dataontosearch is available on PyPI as <https://pypi.python.org/pypi/ckanext-dataontosearch>. To publish a new version to PyPI follow these steps:

1. Update the version number in the `setup.py` file. See PEP 440 for how to choose version numbers, using the principles of semantic versioning.

2. Create a source distribution of the new version:

```
python setup.py sdist
```

3. Upload the source distribution to PyPI (assuming you have run `pip install twine` before):

```
twine upload dist/*
```

4. Tag the new release of the project on GitHub with the version number from the `setup.py` file. For example if the version number in `setup.py` is 0.0.2 then do:

```
git tag 0.0.2
git push --tags
```

Appendix D

Code used for search

This appendix includes a selection of Python code which demonstrates the implementation of the search process. There may be some differences from the code found in the repository, since some irrelevant code is removed, some comments have been put into one line to facilitate easier reading and some methods have been reordered. Some of this code has remained from DataOntoSearch version 1 and is therefore written by Hagelien.

D.1 The OpenDataSemanticFramework class

This class is responsible for constructing matrices and perform searching. The matrix construction code is left out. The class has some style and architectural problems and is a prime candidate for further refactoring; I simply could not dedicate more time to refactoring and therefore changed as little as I could with this class from Hagelien's version.

```
import itertools
from otd.constants import SIMTYPE_AUTOTAG,
    ↪ SIMTYPE_SIMILARITY
import logging
from rdflib import URIRef
from utils.graph import RDF, OTD, DCAT, DCT
from otd.skosnavigate import SKOSNavigate
from otd.queryextractor import QueryExtractor
from otd.semscore import SemScore
import db.dataframe
import db.graph
from sklearn.metrics.pairwise import cosine_similarity
from collections import namedtuple

import pandas as pd
import numpy as np
```

```

# ...

log = logging.getLogger(__name__)

DatasetInfo = namedtuple('DatasetInfo', ('title',
    ↪ 'description', 'uri', 'href'))
SearchResult = namedtuple('SearchResult', ('score', 'info',
    ↪ 'concepts'))
# Note: 'info' is just dataset RDF IRI when dataset_info is
    ↪ disabled
ConceptSimilarity = namedtuple(
    'ConceptSimilarity',
    ('uri', 'label', 'similarity')
)

class OpenDataSemanticFramework:
    def __init__(self, ontology_uuid, dataset_uuid,
        ↪ auto_compute=True,
            concept_similarity=0.0):
        """
        ↪ The RDF library allows a set of rdf-files to be
        ↪ parsed into a graph representing RDF triples. The
        ↪ SKOSNavigate class is a tool for navigating between
        ↪ siblings, children and parents in a graph, and
        ↪ implements methods for calculating similarity based on
        ↪ the relative position of two concepts.
        """
        self.auto_compute = auto_compute
        self.cds = dict()
        self.cds_df_id = dict()
        self.ccs = None
        self.concept_similarity = concept_similarity

        # Set up variables needed for graph property setter
        self.__graph = None
        self.navigator = None
        self.concepts = []
        self.ontology = None
        self.dataset = None
        self._qe = QueryExtractor()
        self._semscore = None

        # Then set graph
        self.load_new_graph(ontology_uuid)

```

```

    # Other properties
    self.dataset_graph =
        ↳ db.graph.Dataset.from_uuid(dataset_uuid).graph

# ...

def search_query(
    self,
    query,
    cds_name="all",
    qc_sim_threshold=0.0,
    score_threshold=0.75,
    include_dataset_info=True,
    include_concepts=True,
):
    """
    Perform a search query.

    Args:
        query: Search query to use.
        cds_name: Name of concept-dataset tagging to
↳ use when retrieving datasets.
        qc_sim_threshold: Lower threshold for how
↳ similar a word in the query must be to a concept label
↳ in order for that concept to be considered relevant to
↳ the query.
        score_threshold: Lower threshold for how
↳ similar a dataset must be to the query to be included
↳ in the result. This effectively decides how many
↳ datasets are included.
        include_dataset_info: Set to False to disable
↳ collection of dataset information. Should save some
↳ time in situations where that information is not
↳ needed.
        include_concepts: Set to False to disable
↳ collection of concepts related to the query and to each
↳ returned dataset. Should save some time in situations
↳ where that information is not needed.

    Returns:
        A tuple. The first item is a list of
↳ SearchResult that matched, sorted with the most similar
↳ results first. The second item is a list of the top
↳ five concepts that were matched with the query.
    """

```

```

log.info('Binding query to concepts...')
# Calculate the query's similarity to our concepts
query_concept_sim =
    → self.calculate_query_sim_to_concepts(
        query,
        qc_sim_threshold
    )

log.info('Extracting most similar concepts...')
# What were the most similar concepts?
if include_concepts:
    most_similar_concepts =
        → self.sort_concept_similarities(
            self.get_concept_similarities_for_query(
                query_concept_sim
            )
        )[:5]
else:
    most_similar_concepts = []

log.info('Comparing datasets to the concepts
    → extracted from the query...')
# How similar are the datasets' similarity to the
    → query's similarity?
dataset_query_sim =
    → self.calculate_dataset_query_sim(
        query_concept_sim,
        cds_name,
        query,
    )

log.info('Putting together information for the
    → result...')
# Put together information for the search results
    → page
results = list()
for dataset, similarity in
    → dataset_query_sim.items():
    # Only consider the most relevant datasets
    # TODO: Return a minimum amount of datasets
    if similarity < float(score_threshold):
        continue
    results.append(SearchResult(
        score=similarity,
        info=self.get_dataset_info(dataset)
    ))

```

```

        if include_dataset_info else dataset,
        concepts=self.get_most_similar_concepts_
        ↪ for_dataset(
            cds_name,
            dataset
        ) if include_concepts else [],
    ))
log.info('Done with query processing!')
return results, most_similar_concepts

def calculate_query_sim_to_concepts(self, query,
    ↪ sim_threshold):
    scorevec = self._semscore.score_vector(query,
    ↪ sim_threshold)
    scorevec = self.enrich_query_with_ccs(scorevec,
    ↪ query, self.concept_similarity)
    return scorevec

def enrich_query_with_ccs(self, score_vec, query,
    ↪ similarity_threshold):
    new_score_vec = score_vec.copy()
    query_row = score_vec.loc[query]
    new_query_row = new_score_vec.loc[query]

    for c1, c2 in itertools.permutations(self.concepts,
    ↪ 2):
        simscore = (float)(self.ccs[c1][c2]) *
        ↪ query_row[c1]

        if simscore < similarity_threshold:
            simscore = 0.0

        new_query_row[c2] = np.nanmax(
            [simscore, new_query_row[c2]]
        )
    return new_score_vec

@staticmethod
def sort_concept_similarities(similarities):
    return sorted(
        similarities,
        key=lambda x: x.similarity,
        reverse=True
    )

```

```

def get_concept_similarities_for_query(self,
    ↪ query_concept_similarity):
    return self._create_concept_similarities(
        query_concept_similarity.columns,
        query_concept_similarity.values[0]
    )

def _create_concept_similarities(self, concepts,
    ↪ similarity_scores):
    processed_similarities = []

    concept_similarities = zip(
        concepts,
        similarity_scores,
    )

    for concept, similarity in concept_similarities:
        labels = self.navigator.pref_and_alt_
        ↪ labels(URIRef(concept))
        label = labels[0]

        processed_
        ↪ similarities.append(ConceptSimilarity(
            concept, label, similarity
        ))

    return processed_similarities

def calculate_dataset_query_sim(self,
    ↪ query_concept_sim, cds_name, query):
    # Add in the datasets' similarity to the concepts
    entry_concept_sim = query_concept_sim.append(
        self.cds[cds_name],
        sort=True
    )

    # Do the similarity calculation
    sim_data = cosine_similarity(entry_concept_sim)

    # Convert into format used by the rest of the
    ↪ application
    entry_entry_sim = pd.DataFrame(
        sim_data,
        columns=entry_concept_sim.index,
        index=entry_concept_sim.index

```

```

)

# Extract the datasets' similarity to the query,
→ sort most similar
# datasets first, then remove the query's
→ similarity to itself
dataset_query_sim = entry_entry_sim \
    .loc[query] \
    .sort_values(ascending=False) \
    .drop(query)
return dataset_query_sim

def get_dataset_info(self, dataset):
    title = next(self.dataset_graph.objects(dataset,
→ DCT.title), None)
    description = next(
        self.dataset_graph.objects(dataset,
→ DCT.description),
        None
    )
    href = next(
        self.dataset_graph.objects(dataset,
→ DCAT.landingPage),
        dataset
    )
    return DatasetInfo(
        str(title),
        str(description),
        str(dataset),
        str(href)
    )

def get_most_similar_concepts_for_dataset(self,
→ cds_name, dataset):
    concepts_for_dataset =
→ self.get_concepts_for_dataset(
        cds_name,
        dataset
    )
    closest_concepts_for_dataset =
→ self.sort_concept_similarities(
        concepts_for_dataset
    )[:5]
    return closest_concepts_for_dataset

```

```

def get_concepts_for_dataset(self, cds_name, dataset):
    location = self.cds[cds_name].loc[dataset]
    return self._create_concept_similarities(
        location.index,
        location.values
    )

```

D.2 QueryExtractor

This class is responsible for taking text and return pairs of tokens and their part of speech. It essentially puts together the NLP methods used in DataOntoSearch. It was initially used only for handling the query, hence its name. Just like with the OpenDataSemanticFramework, this was initially written by Hagelien and modified further by me. It could use some refactoring, e.g. trying to remove the chunk parser.

```

import itertools
import nltk

class QueryExtractor:
    def __init__(self):
        chunk_gram = r"""
            NBAR:
                {<NN.*|JJ.*>*<NN.*>} # Nouns and
→ Adjectives, terminated with Nouns

            NP:
                {<NBAR><IN><NBAR>} # Above, connected with
→ in/of/etc...
                {<NBAR>}
                {<FW|VB.*|JJ.*>} # Experiment, trying to
→ match more
            """
        self.chunk_parser = nltk.RegexpParser(chunk_gram)

    def normalize(self, word):
        return word.lower()

    def extract_terms(self, sentence):
        tokenized_sentence = nltk.word_tokenize(sentence)

        num_tokens = len(tokenized_sentence)
        if num_tokens == 0:
            # Short circuit
            return tuple()

```

```

elif num_tokens == 1:
    # Let WordNet use any part of speech
    token = tokenized_sentence[0]
    return [(self.normalize(token), '')]

pos_tag_tokens =
    ↪ nltk.tag.pos_tag(tokenized_sentence)
tree = self.chunk_parser.parse(pos_tag_tokens)
np_trees = tree.subtrees(filter=lambda t: t.label()
    ↪ == 'NP')
leaves_per_tree = (t.leaves() for t in np_trees)
leaves =
    ↪ itertools.chain.from_iterable(leaves_per_tree)
words_per_leaf = ((self.normalize(w), pos)
    for w, pos in leaves)
return words_per_leaf

def search(self, sentence):
    # TODO: Use extract_terms() directly instead of
    ↪ through this
    return self.extract_terms(sentence)

```

D.3 SemScore

The SemScore class is responsible for taking a query and an ontology navigator (for retrieving concepts), and give back a Query-Concept Similarity Vector. This is where the WordNet synsets are used. Very little code remains from Hagelien's implementation.

```

import itertools
import statistics
import pandas as pd
from nltk.corpus import wordnet as wn

class SemScore:
    def __init__(self, extractor, navigator):
        self.extractor = extractor
        self.navigator = navigator
        self._synsets_by_concept = dict()

    def score_vector(self, query, sim_threshold):
        concepts = list(self.navigator.concepts())
        scoreDataFrame = pd.DataFrame(columns=concepts)
        scoreDataFrame.loc[query] = [0]*len(concepts)

```

```

query_synsets = self.synsets_from_query(query)

for concept in concepts:
    labels = self.synset_sets_from_concept(concept)

    label_scores =
        ↪ [SemScore.calculate_score_for_label(l,
        ↪ query_synsets)
           for l in labels]

    concept_score = max(filter(None, label_scores),
        ↪ default=0.0)
    if concept_score < sim_threshold:
        concept_score = 0.0

    scoreDataFrame.loc[query][concept] =
        ↪ concept_score

return scoreDataFrame

@staticmethod
def calculate_score_for_label(label_words,
    ↪ query_synsets):
    scores =
        ↪ [SemScore.calculate_score_for_label_word(w,
        ↪ query_synsets)
           for w in label_words]
    try:
        return statistics.harmonic_mean(filter(None,
        ↪ scores))
    except statistics.StatisticsError:
        return 0.0

@staticmethod
def calculate_score_for_label_word(label_synsets,
    ↪ query_synsets):
    synset_pairs = itertools.product(query_synsets,
        ↪ label_synsets)
    scores = [q.wup_similarity(c) for q, c in
        ↪ synset_pairs]
    return max(filter(None, scores), default=0.0)

def synsets_from_query(self, q):
    return self.synsets_from_str(q)

```

```

def synset_sets_from_concept(self, c):
    try:
        return self._synsets_by_concept[c]
    except KeyError:
        label_synsets = []
        for label in
            self.navigator.pref_and_alt_labels(c):
                label_synsets.append(self.synset_sets_
                    from_str(label))
        self._synsets_by_concept[c] = label_synsets
        return label_synsets

def synsets_from_str(self, s):
    synset_sets = self.synset_sets_from_str(s)
    return tuple(itertools.chain.from_iterable(synset_
        sets))

def synset_sets_from_str(self, s):
    words = tuple(self.extractor.search(s))
    return self.synset_sets_from_words(words)

def synset_sets_from_words(self, words):
    words = list(words)
    synset_sets = []

    def handle_grams(i):
        for gram_contents in SemScore.ngrams(words, i):
            # Don't construct connected words that
            # didn't exist in the original text. We
            # do this by using placeholders where
            # used words used to be. Skip if we have
            # one of those.
            if (None, None) in gram_contents:
                continue
            gram_words = tuple(word for word, pos in
                gram_contents)
            combined = '_'.join(gram_words)
            synsets = self.synsets(combined, '')
            if synsets:
                # These words have been matched, don't
                # look up afterwards
                for tagged_word in gram_contents:
                    try:
                        index =
                            words.index(tagged_word)

```

```

        # TODO: Pass indices instead of
        → contents from ngrams, so
        → array updates here are
        → reflected in the next
        → iteration while still
        → inside of handle_grams()
        words[index] = (None, None)
    except ValueError:
        # ValueError triggered by
        → words.index; not found. The
        → word was probably removed
        → by an earlier ngram with
        → the same n, so we don't
        → need to remove it again
        pass

    # Add the result
    synset_sets.append(synsets)

handle_grams(3)
handle_grams(2)

# Lookup any remaining single-words
for word, pos in words:
    if word is not None:
        synset_sets.append(self.synsets(word, pos))

return synset_sets

@staticmethod
def ngrams(iterator, n):
    iterator = tuple(iterator)

    def get_ngram_at(offset):
        return tuple(iterator[offset + i] for i in
            → range(n))

    return tuple(get_ngram_at(i) for i in range(0,
        → (len(iterator) - n) + 1))

def synsets(self, word, pos):
    return wn.synsets(word,
        → pos=self.convert_to_wn_pos(pos))

def convert_to_wn_pos(self, pos):

```

```
if pos.startswith('NN'):
    return wn.NOUN
elif pos.startswith('VB'):
    return wn.VERB
elif pos.startswith('JJ'):
    return wn.ADJ
elif pos.startswith('RB'):
    return wn.ADV
else:
    return None
```

