

Obligatory assignment INFO116

Candidate numbers: 145, 147, 192



Introduction

In this assignment we were given a task that was about giving meaning to the philosophy papers written by Wittgenstein and some commentary articles that reflects on what Wittgenstein is talking about. After reading these articles, the task was to make some sense and meaning of what Wittgenstein's paper is really about. There seems to be many different ways to make sense out of philosophy, as there are no direct answer to what is right and what is wrong. This is where the commentary articles came in handy with some different views.

Once we had read both the target article written by Wittgenstein and the commentary articles, we started making an ontology using protégé. The creation and use of the ontology is described later on in this report. However, because there was no specific answer to how this should be done right, it could be considered the most difficult part of this assignment. We still managed to create an ontology that made some sense.

After we had created this ontology we downloaded the html source code of the target article and the commentary articles, then we started annotating the source code using RDFa and JSON-LD.

Finally, in the end we created some SPARQL queries to fetch some information from the texts.

The owl ontology, the annotated html pages and the SPARQL queries are also delivered as separate files together with this written report. They are still described in more detail throughout this report.

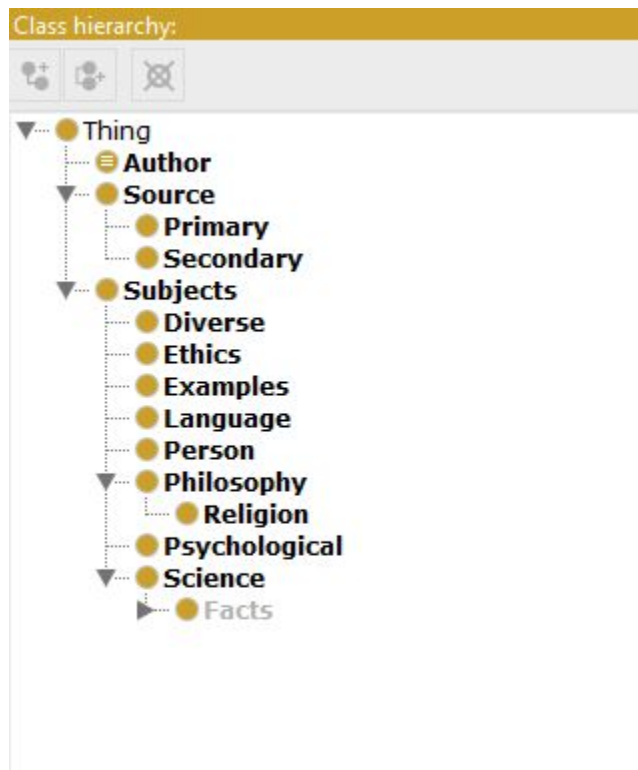
Ontology

This part of the assignment is to create an ontology for the work of Wittgenstein, along with 3 papers discussing his work. The main goal is to try find a way to represent the papers in a way we can annotate and retrieve data semantically from the text. This is related to the efforts to digitalize his works and make them public.

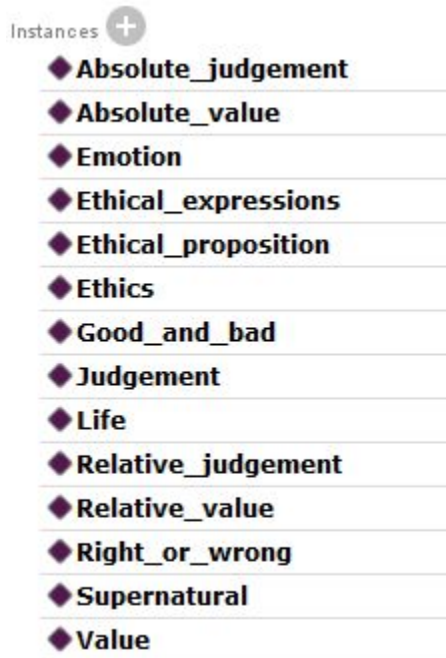
When we first started working on creating the owl file we began reading the articles. First we read the Target article written by Wittgenstein himself and then we read complementary articles that discussed what the article was about. We printed out these texts and actually read them on paper so we could easier mark the important

and relevant information by simply using a marker pen. We felt this was an easier way to find the relevant information from the text instead of just reading it.

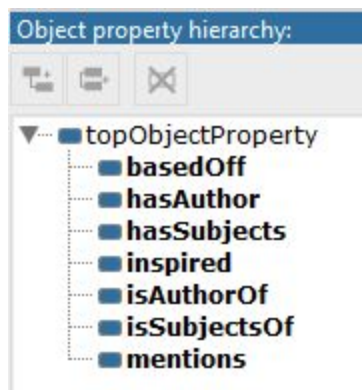
When we had finished reading and marking up the texts we opened up protege and started creating an ontology. We based the ontology on what we thought would be relevant from the text and placed this information in under different classes to create a somewhat reasonable ontology that made sense. We have included a picture of the result we ended up with using protege below:



As you can see in the picture this is what the classes look like. In addition to these classes we also added members under most of the subclasses as well. These members can also be referred to as instances. An example of the members on one of the subclasses is shown below:



This picture above, is showing the members or instances of the class “Ethics” which is a subclass of subjects. To connect these members together we created some object notations such as hasAuthor and the reverse isAuthorOf.



^ These are object notations we included in the ontology. These connect the different classes and members together to create some sort of structure to the ontology. A demonstration of how this works is shown in pictures below:

Description: Author

Equivalent To 

● {Deirdre_Christine_Page_Smith , Janyne_Sattler , Ludwig_Wittgenstein , Paul_Formosa}

SubClass Of 

● isAuthorOf min 1 Source

General class axioms 

SubClass Of (Anonymous Ancestor)

Instances 

- ◆ Deirdre_Christine_Page_Smith
- ◆ Janyne_Sattler
- ◆ Ludwig_Wittgenstein
- ◆ Paul_Formosa

In the picture above the text marked inside the bubble explains how the Author class isAuthor of minimum 1 source. The inverse, hasAuthor, is then shown in the Source class.

Description: Source


Equivalent To 

SubClass Of 

- hasAuthor min 1 Author
- hasSubjects min 1 Subjects

The inverse of isAuthorOf is hasAuthor and is shown in the picture above from the Source class. In addition to having an author, the source also have minimum 1 subject aswell.

Property assertions: Formosa_text

Object property assertions 

- mentions Bad
- mentions Religion
- basedOff Wittgenstein_text
- mentions Metaphor
- mentions Judgement
- mentions Facts
- mentions Book
- mentions World
- mentions Value

The picture above is showing some property assertions of the Formosa_text, which is one of the members of the subclass Secondary that has the superclass Source. These assertions are mostly about what Formosa mentions in the text and also what he is basing the information on. Such as in the example picture above where it says “basedOff Wittgenstein_text”. The mentions property means basically what is says, the word is mentioned in the text. This have also been done to the other sources

Creating this ontology helped us get a better overview and understanding of Wittgenstein’s “Lecture of ethics” and its commentary texts. We could then use these words from the ontology to annotate the source of all the article’s html pages. The annotation is shown in the next step of this assignment.

We did some initial attempts at topic extraction using Python and scikit-learn. This was a rather interesting task as we had some hopes that this could lend us some help getting the important words out. This turned out to be a rather poor idea, as extracting topics and context from a philosophical text is amazingly hard, and since we decided that we didn’t want to accomplish a PhD in 2 months we moved quite quickly on. It is however quite interesting to look at the resulting topics from the texts.

The script can be found inside `Python/topics.py` and for the sake of the report, i’ll be running with the top 6 words, the script itself will list all the candidate words.

Main text

- Topic 0: say let right believe make thing fact
- Topic 1: fact way say don words certainly think
- Topic 2: right say don believe thing words certainly
- Topic 3: make believe don words certainly think way
- Topic 4: words way think thing say right make

Sorites text

- Topic 0: world wittgenstein foot note follows sense way
- Topic 1: ethical wittgenstein sense propositions note foot state
- Topic 2: nonsense propositions sense state wittgenstein logical note
- Topic 3: wittgenstein tractatus ethical nonsense note foot thesis
- Topic 4: wittgenstein thesis world note foot ethical way

Rediscover text

- Topic 0: wittgenstein semantic 139a ms philosophy examples labelling
- Topic 1: wab wittgenstein texts nachlass project examples work
- Topic 2: labelling semantic ms 139a make text standard
- Topic 3: discovery texts wab wittgenstein philosophical examples machine
- Topic 4: labelling examples semantic text goal use bee
- Topic 5: use labelling philosophical wittgenstein examples sense philosophy
- Topic 6: wab standard texts text nachlass project work
- Topic 7: ms standard text 139a labelling make semantic

Topic 8: edition nachlass texts semantic wittgenstein work different
 Topic 9: philosophy 139a ms use ethics work process
 Topic 10: edition readable machine nachlass texts project work
 Topic 11: semantic goal make philosophical labelling text examples
 Topic 12: wab work nachlass project using ethics lecture
 Topic 13: texts ethics lecture bee make sense semantic
 Topic 14: bee lecture ethics 139a discovery philosophical using
 Topic 15: nachlass texts semantic work bee different process
 Topic 16: discovery semantic text work ms bee different
 Topic 17: wittgenstein project nachlass wab goal work machine
 Topic 18: bee text process using standard work different
 Topic 19: philosophical text philosophy ms use 139a sense
 Topic 20: different wittgenstein nachlass wab machine readable ethics
 Topic 21: work ethics 139a ms lecture using different
 Topic 22: philosophical texts goal project discovery wab wittgenstein
 Topic 23: project philosophy wittgenstein readable machine nachlass ms
 Topic 24: edition bee semantic work machine readable ms
 Topic 25: wittgenstein nachlass machine readable work ms bee
 Topic 26: wittgenstein work nachlass bee different discovery edition
 Topic 27: bee wab different wittgenstein readable machine nachlass
 Topic 28: texts nachlass process different work semantic bee
 Topic 29: text machine readable work ms bee different
 Topic 30: bee different wittgenstein nachlass wab work ms

The "Main Text" is wittgenstein and the other is two of the supporting texts.

The main thing i could draw from this is that it is quite hard to get anything sensical out of Wittgensteins text. However, interestingly, we do notice a little pattern of recurring topics in the side text we where suppose to study. However, this was far from sufficient to create an ontology and was probably a path way over our head.

The html code is essentially stripped away because it was easier this way. It is then split up into sections so we can isolate the different sections of the text. The code is essentially piggybacking greatly on scikit-learn incredible library for machine learning. The code essentially stuffs the text sections into the CountVectorizer to do basic feature extraction based on word frequency. We then use the decomposition.NMF class to, if i understand it correctly, get the individual scores of the words as possible topics. Rest is essentially vector transformation on the matrixes.

It was an interesting task and lead to the future code work in the task.

Annotation using RDFa and JSON-LD

This is essentially where things became interesting. Instead of annotating all the interesting topics by hand, we decided to try and make a script. This turned out to not be a horribly difficult and time consuming task. With a few hours work we were able to annotate all the member words in our Subjects with enough accuracy to work with. There are probably some edge cases that we simply haven't spotted. But i believe this demonstrate the work pretty well.

Essentially what we do is that we have words like "absolute values" cased in the ontology as "Absolute_value", this was a premature decision we stuck with before we realized this could be tricky to parse, and choose one word. In this case "value", then we look behind for "Absolute" then forward "of". This would turn into "Absolute_value_of". This would result in an empty hit in our ontology, so we move on to only word and "backward"; "Absolute_value". This would turn up positive in our ontology and get annotated with "Ethics" and placed back into our list. The removed word is replaced with a "None" and removed at a later step. We then finish off by concatenating our list of lists and prepending a span for our typeof and thus sealing the deal.

This demonstrates the transformation from text to annotated text.

Raw data:

```
value Value
absolute value
<asdfsas>value something
heyho Value Absolute
right or wrong
```

Place into lists:

```
[['value', 'Value'],
 ['absolute', 'value'],
 ['<asdfsas>value', 'something'],
 ['heyho', 'Value', 'Absolute'],
 ['right', 'or', 'wrong']]
```

Annotate:

```
[['<span property="Ethics">value</span>', '<span property="Ethics">Value</span>'],
 [None, '<span property="Ethics">absolute value</span>', None],
 ['<asdfsas>value', 'something'],
 ['heyho', '<span property="Ethics">Value</span>', 'Absolute'],
 [None, '<span property="Ethics">right or wrong</span>', None]]
```

Join together the lists to form new text and remove None:

```
<span property="Ethics">value</span> <span property="Ethics">Value</span>
<span property="Ethics">absolute value</span>
<asdfsas>value something
```


heyho Value Absolute
right or wrong

Since we do filter out used and unused word in our ontology script we were able to cheat off this later during the JSON-LD annotation which we will explain later, along with creating an interesting enough ontology for our SPARQL queries. In the example you also see one of the edge-cases we did not have time to solve, as the html tags wasn't something we spotted before later, and didn't have time enough to remove. The next step was to essentially patch up the missing annotations into the html. This wound up only being the authors names as the rest of the annotations seems good enough.

Since we stored the used topics for each webpage, we could generate much of the topics (mentions) of our pages by the script. This made json-ld much easier to translate from RDF and the OWL ontology. Rest was just plugging the expected variables into the page.

```
{
  "@context": {
    "uib": "http://example.org/index.rdf#",
    "owl": "http://www.w3.org/2002/07/owl#"
  },
  "@type": [
    "http://www.w3.org/2002/07/owl#NamedIndividual",
    "wit:Source"
  ],
  "wit:hasAuthor": {
    "@id": "Ludwig_Wittgenstein"
  },
  "wit:Primary": {
    "@id": "Wittgenstein_text"
  },
  "wit:mentions": [
    {"@id": "wit:Relative_value"},
    {"@id": "wit:Lecture"},
    {"@id": "wit:Bad"},
    {"@id": "wit:Psychological"},
    {"@id": "wit:Sense"},
    {"@id": "wit:Science"},
    {"@id": "wit:Metaphor"},
    {"@id": "wit:Emotion"},
    {"@id": "wit:Facts"},
    {"@id": "wit:Person"},
    {"@id": "wit:Scientific"},
    {"@id": "wit:Religious"},
    {"@id": "wit:Expression"},
    {"@id": "wit:Religion"},
  ]
}
```

```

{"@id": "wit:World"},
{"@id": "wit:Ethics"},
{"@id": "wit:Value"},
{"@id": "wit:Words"},
{"@id": "wit:Absolute_value"},
{"@id": "wit:Supernatural"},
{"@id": "wit:Hamlet"},
{"@id": "wit:Logic"},
{"@id": "wit:Life"},
{"@id": "wit:Book"},
{"@id": "wit:I_wonder"},
{"@id": "wit:Definition"},
{"@id": "wit:Language"},
{"@id": "wit:Meaning"},
{"@id": "wit:Good"}
]
}

```

This was pretty much what we did for our annotations. It worked pretty great and we didn't really encounter any technical debts taking this route to solve the task.

Notes about SPARQL

The SPARQL part of this was more trickier than expected. Our assumption was that we could somehow merge the annotated pages together using the RDFa from the <http://rdfa.info> site, but this failed horrible.

What we did instead was to again fetch all the topic mentions from the text, but formatted it into RDF/XML and added them into the existing ontology by hand. Essentially just copy-pasting 3000 lines of XML that the script generated for us.

```

<owl:NamedIndividual rdf:about="http://example.org/index.rdf#Wittgenstein_text">
  <rdf:type rdf:resource="http://example.org/index.rdf#Primary"/>
  <mentions rdf:resource="http://example.org/index.rdf#Life"/>
</owl:NamedIndividual>

```

This is the output, times the amount of texts and topics found. We insert the name of the Text, either if the source is Primary or Secondary and the Subject we have. It is a very verbose syntax so the resulting file turns out pretty big. But the amount of added code was barely 15 lines to generate this. Big time saver.

We could then load this up along with our ontology and get better queries without any extra work for us. Located inside the “sparql.txt” file.

Contribution and workflow

The members of our group are: Jellyfish, Fox and Goat.

All the members of the group had somewhat different assignments. We tried to split it so that we would save as much time as possible by not having everyone read the same texts at the same time. Our member Jellyfish started off by reading Wittgenstein’s paper on ethics, which resulted in even more confusion and Fox and Goat took the other 2 commentary articles. The papers were filled with nonsense and philosophical questions that didn’t really affect the assignment. Eventually we decided to print out the papers to start marking what we thought was relevant words and sentences that could somehow give a better understanding of what the texts were about and possibly make some sense of it. After having a gigantic list of words and expressions, we started making bubbles on a sheet, separating words into themes so that it would be easier to implement into the ontology.

When we started to make the ontology it was hard to understand Pròtege. Goat and Jellyfish started going through the tutorials over again and looking at different examples such as the wine.owl and pizza.owl files. These ontologies had quite different design. For instance the pizza ontology had a lot more classes compared to the wine ontology that used a lot more members/instances instead. Once we figured out how this worked, we decided to create members under the different classes as this seemed more fitting to our situation.

Once we had more control of how the tool worked, we started to implement the classes and structure of the ontology as we had sketched on paper using a basic tree structured like mindmap. The classes described themes, and members were instances of the themes. Once creation of the classes and subclasses was done, Goat started making the object properties to connect the different classes to each other.

After ‘completing’ the ontology we started looking at annotating into html. Jellyfish and Goat started annotating one of the html pages each, but Fox however, had a different plan. Instead of annotating these html pages manually by inserting spans for every word related to the ontology, Fox wrote a script that checked every single word in the texts to check whether they were in the ontology or not. If the words were in the ontology the script would automatically mark them. Because of Fox’s amazing script saved the group a lot of time compared to what it would have taken us if we were to do it all manually. This turned out to be most of the work we did in our ontology and continues looking at SPARQL

The SPARQL itself was kinda tricky. One of us works with SQL, but the others have touched upon SQL before, so SPARQL was pretty hard to get our head around. We do believe we returned with a few interesting queries, but we sadly ran out of time to try and get out any

more complex queries up and running to try and compare the different texts with topics and words.

Throughout this assignment we have learned how the use of an ontology could be used to annotate and create semantic meaning to texts that doesn't even make much sense to a computer, such as philosophy. The script we used to to annotate the html pages helped us save some time. In addition to this, we learned how to fetch data using sparql queries. All in all this assignment turned to give give us a lot more knowledge than expected before we started working.

And thus ends the adventure of Jellyfish, Fox and Goat!