Universität Zürich, Philosophische Fakultät

Zentralbibliothek Zürich, MAS Bibliotheks- und
Informationswissenschaft

Machine indexing of institutional repositories: indexing Edoc with Annif as proof of concept

🆔 Dr. Maximilian G. Hindermann
Spalentorweg 46, CH-4051 Basel
+41 79 277 0612
maximilian.hindermann@gmail.com

🆔 Silke Bellanger, Referentin

🆔 Dr. Andreas Ledl, Ko-Referent

1. April 2021

Abstract

Lorem ipsum.

Contents

9 Bibliography 45

1 Introduction

In this section I will. . .

1.1 Problem statement and goal

Edoc is the institutional repository of the University of Basel. It runs on the EPrints 3 document management system (https://github.com/eprints/eprints). Edoc was conceived in 2009 as repository for electronic dissertations and grew in scope when the University of Basel adapted its first open access policy in 2013. Today Edoc contains roughly 68'000 items, most of them journal articles without full text support. Even though descriptive and administrative metadata are collected for each item, this does not include subject terms. The lack of subject indexing hampers the repository's usability and potential with respect to the topic-based monitoring of research output and, to a lesser extent, information retrieval. The goal of this project is to remedy this situation with the help of machine indexing.

!! More precisely, this project aims to provide a roadmap for indexing the elements of Edoc with Annif (Suominen, 2018) – an open source tool for automated subject indexing and classification – and BARTOC FAST (Hindermann and Ledl, 2020), a federated asynchronous search tool that covers concepts of thousands of knowledge organization systems like thesauri, classifications etc.

!! The goal of this thesis is to investigate to what extent machine indexing can be used to ameliorate institutional repositories.

1.2 Method

!! Say something to the effect that all data and code are available on GitHub.

!! In this chapter, the prototype for a machine indexing of Edoc is presented. The focus is primarily on practical implementation although care is taken to spell out design decisions in as much detail as is needed.

1.3 Outline

Let us start by stating the aim of the prototype. From a functional perspective, the prototype takes a subset of the data from Edoc as input and provides index terms for each item in this subset as output. In order to fulfill this aim we can distinguish a number of steps that need to be taken:

1. Understand the Edoc data.

2. Select and construct a sample data set.
3. Use Annif to index the items in the sample data set.
4. Construct a gold standard from the keywords.
5. Assess the quality of the output based on the gold standard.

In the sections below, these steps will be discussed in detail.

1.4 Tools

For data refinement I use OpenRefine version 3.4.1 (https://openrefine.org/). Data manipulation in OpenRefine is tracked: the manipulation history of some data can be exported as JSON file and then reproduced (on the same data on a different machine or on different data) by loading said file. I will supply a corresponding manipulation history whenever appropriate. Note that when using OpenRefine with larger files (and there are many such files in this project), the memory allocation to OpenRefine must be manually increased (see https://docs.openrefine.org/manual/installing/#increasing-memory-allocation for details). Also note that OpenRefine always counts the number of records by the first column (a fact which caused me many headaches).

2 Edoc data

In this section, I will describe the Edoc data and how it was extracted.

2.1 Edoc data extraction

Even though Edoc is a public server, its database does not have a web-ready API. In addition, since Edoc is a production server, its underlying database cannot be used directly on pain of disturbing the provided services. The data hence needs to be extracted from Edoc in order to work with it. This could for example be achieved by cloning the database of the server and running it on a local machine. However, this route was not available due to the limited resources of the responsible co-workers. I thus employed a workaround: Edoc has a built-in advanced search tool where the results can be exported. Even though it is not possible to extract all database items by default, only one of the many available search fields has to be filled in. The complete database can hence be extracted by only filling in the

`date` field and using a sufficiently permissive time period such as 1940 to 2020 since the oldest record in the database was published in 1942. On January 21 2021, this query yielded 68'345 results. These results were then exported as a 326 MB JSON file called `1942-2020.json`. `1942-2020.json` has two drawbacks. First, the maximum file size on GitHub is 100 MB. And second, `1942-2020.json` is too big to be handled by standard editors requiring special editors such as Oxygen. For these reasons `1942-2020.json` was split into smaller files that are easier handled and can be uploaded to GitHub. For this `Utility.split_json` was used yielding 14 files of size 20 MB or less containing 5000 or fewer entries each. These files are called `raw_master_{year}.json` and saved under `/files/raw`.

2.2   Data description

Each Edoc item has 43 main data fields. Most of these data fields belong to the class of descriptive metadata. The data entry for an item's descriptive metadata is undertaken by the person who uploads the item which is in most cases one of the item's authors. The Note that an item is first uploaded to the University of Basel's research database (https://www.forschdb2.unibas.ch) before being automatically fed into Edoc. There are explicit rules on how to enter the data as summarized in the user manual (Universität Basel, n.d.). However, there is no (or at least no systematic) manual or automatic validation of the entered data. This means that some data fields are very heterogenous and require further parsing (see section "Gold standard"). The data fields that are of interest to the project at hand are discussed in more detail in (see section "Sample data set" below).

3   Sample data set

In this section, I will explain how the sample data set is selected and constructed. "Selection" hereby means the task of specifying a subset of the Edoc data; by "construction" I mean the task of implementing this selection.

3.1   Selection

There are a number of constraints for selecting the sample data set pertaining to an items's text, evaluability and the data set's scope.

The first constraint stems from Annif, the machine indexing framework that will be used. Put simply, machine indexing assigns index terms to a

text based on training data. The training data consists of a set of texts, a vocabulary, and function from vocabulary to texts. In this context we mean by "text" a sequence of words in a natural language and by "vocabulary" a set of words. Intuitively, the training data consists of pairs of text and subject terms that meet some standard. The training data and the vocabulary are already supplied by Annif, we only need to provide a text for each item that we want to be assigned index terms. Therefore, we only select items with text.

The second constraint is that we must be able to evaluate the quality of the index terms assigned by Annif to any item in the sample data set.

The third and final constraint takes into account the possibility of having to extend the prototype to the complete Edoc data. On the one hand, the sample data set should be small enough to allow for easy handling and rapid iteration. On the other hand, the sample data set should not be trivial but reflect the quirks and inadequacies of the complete dataset. In other words, we are looking for an abstraction rather than an idealization in the sense of Stokhof and van Lambalgen (2011).

3.2   Construction

The first constraint is that each item in the sample data set needs to have a text. Intuitively, this text could consist of a title, an abstract, or even a full text, or any combination of the above. Similar to manual indexing, having more information (that is, longer texts) usually implies a higher indexing quality in machine indexing. However, this assumption needs to be confirmed empirically for the given machine indexing framework and data set. If indeed confirmed, it might be advisable only to index items that have a certain minimal text length. Therefore, in order to construct the sample data set, we require each item to have a non-empty value in the data fields `title`, `abstract`, and `id_number` as proxy to retrieve a full text remotely. Note that even more context could be provided by taking into account other data fields such as `type`, `publication` or `department`. Especially the latter might be valuable when disambiguating homonymous or polysemous words. For example, consider the item `https://edoc.unibas.ch/id/eprint/76510` which is titled "Blacking Out." Without further context, this title could refer (amongst others) to a physiological phenomenon, a sudden loss of electricity, or a measure taken in wartime. Knowing that the item was published by a researcher employed by the Faculty of Business and Economics (and not, say, by the Faculty of Medicine) gives us reason to exclude the first meaning.

However, this idea cannot be implemented with the out of the box instance of Annif employed in this chapter (see section "Machine indexing").

The second constraint is that the index terms assigned to each item in the sample data set must be assessable. How the assessment is conducted in detail is discussed in section "Assessment". For now, it is sufficient to say that we require a standard against which we can evaluate the assigned index terms. By "standard" I mean that given a vocabulary of index terms and our sample data set, for every item in the sample data set, there is an approved subset of the vocabulary. The production of a standard from scratch is of course is very costly since a person, usually highly skilled in a certain academic domain, must assign and/or approve each item's index terms (!!source). For this reason we try to avoid having to produce a standard. One way of doing so is by requiring items in the sample data set to have non-empty `keywords` data field. The caveats of this approach are discussed in section "Assessment".

Taking into account the third constraint simply means that we do not employ any further restrictions. We can hence construct the sample data set by choosing exactly those items from `/files/raw` which have non-empty data fields `title`, `abstract`, `id_number`, and `keywords`. We do so by calling `Data.select_from_file` iteratively for all files in `/files/raw`. The resulting file is saved in `/files/selected` as `selected_master.json`.

3.3 Analysis

Of the 68'345 items in `/files/raw`, all have a title (non-empty `title` data field), a little more than half of the items have an abstract (37'381 items with non-empty `abstract` data field), roughly half of the items have an ID (35'355 items with non-empty `id_number` data field),[1] and less than 10% of the items have keywords (6'660 items with non-empty `keywords` data field). The sample data set as requires all the above data fields to be non-empty; `/files/sample/sample_master.json` has 4'111 items and hence constitutes 6% of the raw data.

In order to determine how well the sample data set represents the raw Edoc data, a one-sample χ^2 goodnes of fit test was conducted on each selection data field (following Parke 2013, chap. 1). The results indicate that the sample data proportions of items are significantly different from the raw

---

[1]Note that when using the facet by blank on `id_number` in OpenRefine, there are 57'153 matches; this is due to the fact that many items with an ID such as DOI have secondary or tertiary IDs such as ISI or PMID.
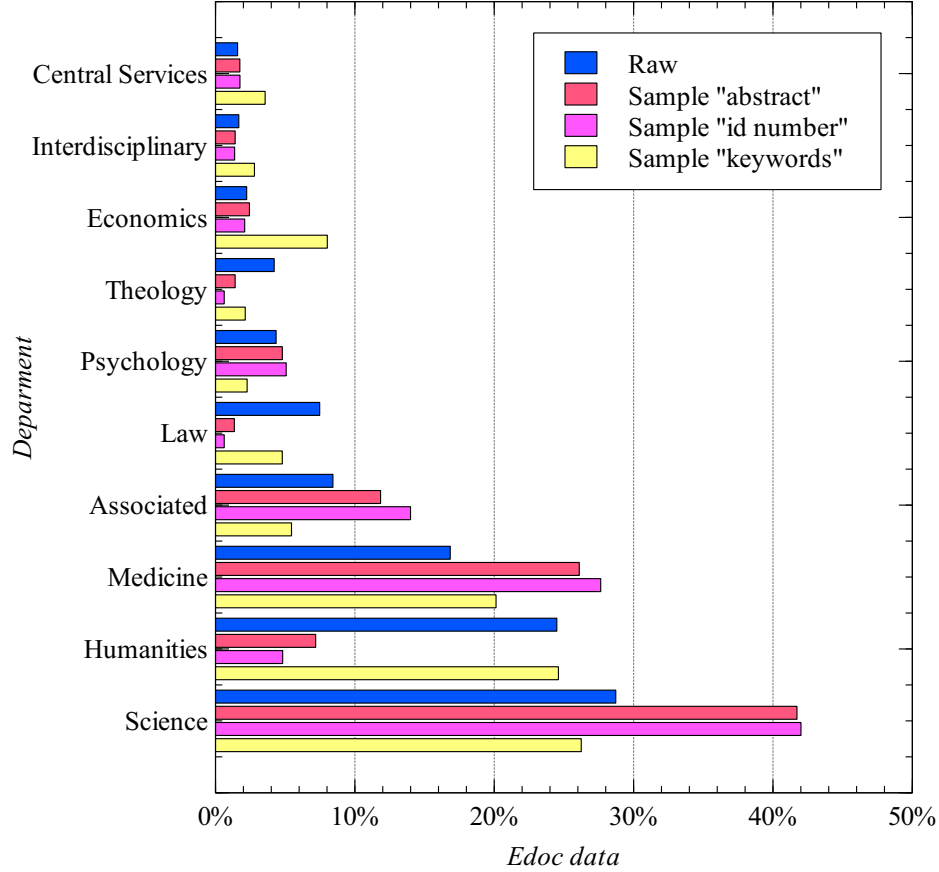
Figure 1: The sample data set ($n = 4$'111) is not representative of the raw Edoc data per department. Data field `abstract`: $\chi^2(df = 9) = 3$'160.556, $p < 0.001$; data field `id_number`: $\chi^2(df = 9) = 4$'209.0285, $p < 0.001$; data field `keywords`: $\chi^2(df = 9) = 2$'314.533, $p < 0.001$. The data foundation is available at `/files/analysis/chi_square_{data field}` and the χ^2-statistic can be calculated by calling `Analysis.print_chi_square_fit` on the data foundation files.

Edoc data per department (see Figure 1 for more details).

The sample data set is hence not representative of the raw Edoc data. This is not surprising since its construction is strongly biased. This bias has the effect that the sample data set is significantly skewed towards English journal publications in the sciences, medicine and economics from the 21st century (again, this is shown by a one-sample χ^2 goodness of fit test, see Figure 2 for more details). The upshot of this analysis is that the humanities are underrepresented in the sample data set. Therefore, any results with respect to the quality of machine indexing discussed below might not be applicable to the humanities.

4 Machine indexing

In this section, I will briefly introduce Annif and how indexing with Annif was implemented. For a general overview concerning machine indexing, see Golub (2019).

4.1 Annif

Annif (Suominen et al. 2021) is an open source multi-algorithm automated indexing tool developed at the Finnish National Library (see Suominen 2019 for an overview). Training a local Annif instance for machine indexing is currently evaluated at different libraries, for example at the German National Library (Nagelschmidt 12.11.2020). As stated in section section "Introduction", in this project I am interested in using Annif out of the box. This means that instead of training a local Annif instance, I will be relying on the Annif REST-style API (see https://api.annif.org/).[2] The advantage of this approach is that it is quick and relatively easy to implement. The disadvantage is that no custom algorithms or vocabularies can be used. The task of constructing a gold standard (see section "Gold standard") remains the same.

Annif vocs that are available

4.2 Implementation

Machine indexing with Annif was implemented using the annif-client Python library (see https://pypi.org/project/annif-client/). The desired

---

[2]This API is specifically intended for testing. For integration into a production system, the Finto AI API is available (see https://ai.finto.fi/v1/ui/).
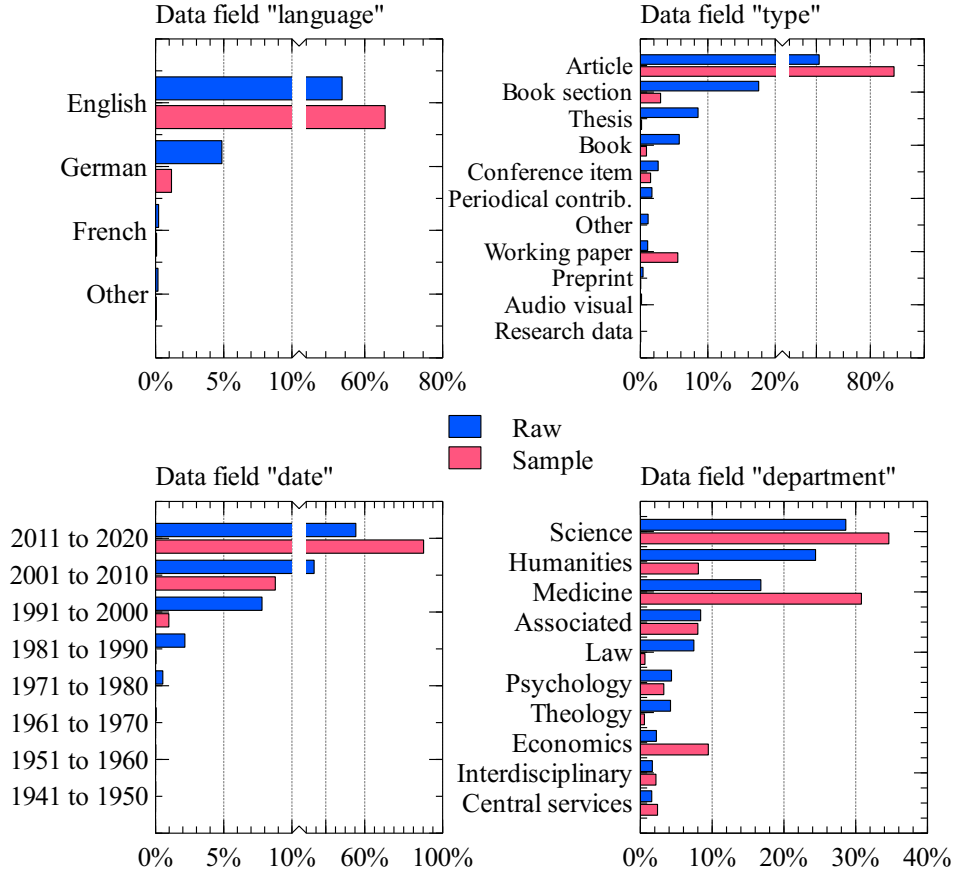
Figure 2: The sample data set ($n = 4'111$) is significantly skewed towards English (data field `language`: $\chi^2(df = 3) = 1'441.414$, $p < 0.001$) journal publications (data field `type`: $\chi^2(df = 10) = 52'519.743$, $p < 0.001$) in the sciences, medicine and economics (data field `department`: $\chi^2(df = 9) = 14'447.276$, $p < 0.001$) from the 21st century (data field `date`: $\chi^2(df = 7) = 35'878.493$, $p < 0.001$) as compared to the raw Edoc data. The data foundation is available at `/files/analysis/chi_square_{data field}` and the χ^2-statistic can be calculated by calling `Analysis.print_chi_square_fit` on the data foundation files.

output is that every item in the Edoc sample data set is assigned subject terms by all available Annif API projects (namely, `yso-en`, `yso-maui-en`, `yso-bonsai-en`, `yso-fasttext-en`, `wikidata-en`) based on all available text bases (title or title and abstract). In order to distinguish these assignments, we use a unique ID (called "marker" in what follows) for each Annif configuration. A marker is constructed by the convention: `{project_id}-{abstract}-{fulltext}-{n}-{threshold}`, where `project_id` is the Annif project, `abstract` and `fulltext` are boolean variables, `n` is the maximum number of suggestions per item, and `threshold` is a variable for a threshold Annif score.

In order to generate the ouput, we call `Data.super_enrich_with_annif`, thereby iteratively calling `Data.enrich_with_annif` for all items in `/files/sample/sample_master.json` for all Annif projects. The output is saved in `/indexed/` as `indexed_master.json`.

5 Gold standard

In this section I will construct a derivative gold standard in order to assess the quality of machine indexing the sample data set with Annif.

5.1 Definition

The most common approach for assessing the output of machine indexing is by systematically comparing it with a gold standard (sometimes also referred to as model or reference). Golub et al. (2016, 10) define a gold standard as "a collection in which each document is assigned a set of [subject terms] that is assumed to be complete and correct" where "*complete* means that all subjects that should be assigned to a document are in fact assigned, and *correct* means that there are no subjects assigned that are irrelevant to the content." Put inversely, if an item in the gold standard lacks subject terms describing its content, the assignment is not complete; and if an item in the gold standard has been assigned subject terms that are not relevant to its content, the assignment is not correct.

A gold standard is usually the product of manual indexing by "information experts, subject experts, or potential or real end users" (Golub et al. 2016, 10). This entails its own host of epistemic problems relating to objectivity and consistency of the assigned subject terms. Most importantly, however, the construction of a gold standard from scratch is very expensive. It is therefore not an option for the project at hand. Rather, I will construct

what could be called a "derivative" gold standard by reusing indexing data that is already available.

There are other methods for assessing machine indexing quality besides comparison with a gold standard that are worth mentioning. These include an evaluation in the context of indexing workflows (Golub et al. 2016, 13ff.), the assessment of retrieval performance (Golub et al. 2016, 15–23.), and so-called model free assessments (Louis and Nenkova 2013).

The construction of the derivative gold standard takes three steps:

1. Extract the keywords from the sample data set.
2. Clean the extracted keywords.
3. Reconcile the extracted keywords with.

The intended output of this transformation process is...

These steps are explained in more detail in what follows.

5.2 Extract and clean keywords

In a first step, the keywords must be extracted from the sample data set `/sample/sample_master.json`. Recall that we mandated a non-empty `keywords` data field for an item to be selected from the raw Edoc data (see subsection "Selection"). We can thus simply copy the information in the `keywords` data field on a per item basis. To do this, we call `Keywords.extract_keywords` with the sample data set as argument and save the output as `keywords/keywords_extracted.json` like so:

```
keywords = Keywords.extract_keywords(DIR + "/sample/sample_master.json")
Utility.save_json(keywords, DIR + "/keywords/keywords_extracted.json")
```

In second step, a list of all single keywords must be created. In order to do so, let us consider now in more detail the exact information extracted from the `keywords` data fields as per `/keywords/keywords_extracted.json`. In Edoc, the `keywords` data field of an item is a non-mandatory free text field that is filled in by the user (usually one of the authors) who undertakes the data entry of an item to Edoc. Even though the Edoc user manual specifies that keywords must be separated by commas (Universität Basel, n.d., 8), this requirement is neither validated by the input mask nor by an administrator of Edoc. Furthermore, neither the manual nor the input mask provide a definition of the term "keyword." A vocabulary or a list of vocabularies from which to choose the keywords is also lacking. Taken together, these

observations are indicative of very heterogeneous data in the `keywords` data field. To wit, the items of `/keywords/keywords_extracted.json` are strings where single keywords are individuated by any symbols the user saw fit. So, for each item in `/keywords/keywords_extracted.json`, the user input must be parsed into single keywords.

Next the so parsed single keywords must be cleaned or normalized: we want the keywords to follow a uniform format thereby joining morphological duplicates such as "Gene," "gene," "gene_," "gene/," and so on. Also, some keywords are in fact keyword chains, for example "Dendrites/metabolism/ultrastructure." Keyword chains must be broken into their component keywords and then parsed again. The reason for this is that Annif only assigns flat keywords and not keyword chains.

`Keywords.clean_keywords` is the implementation the parser while the recursive cleaner is implemented by `Keywords.clean_keyword`. To create the desired list of clean keywords, saved as `/keywords/keywords_clean.json`, we call `Keywords.clean_keywords` with the extracted keywords as argument:

```
keywords_extracted = Utility.load_json(DIR + "/keywords/keywords_extracted.json")
keywords_clean = Keywords.clean_keywords(keywords_extracted)
Utility.save_json(keywords_clean, DIR + "keywords/keywords_clean.json")
```

5.3 Analysis

`/keywords/keywords_clean.json` has 36'901 entries many of which are duplicates. We hence deduplicate and count the number of occurrences of each keyword. This is achieved by calling `Keywords.make_histogram` on `/keywords/keywords_clean.json` and saving the output as `/keywords/keywords_clean_histogram.json`:

```
keywords_clean = Utility.load_json(DIR + "/keywords/keywords_clean.json")
keywords_histogram = Keywords.make_histogram(keywords_clean)
Utility.save_json(keywords_histogram, DIR + "keywords/keywords_clean_histogram.json")
```

An analysis of `/keywords/keywords_clean_histogram.json` shows that the lion's share of keywords has only one occurrence but that the total occurrences are predominantly made up of keywords with more than one occurrence (see Figure 3 for details).

To analyse the spread of keywords in the sample data set, the keywords per item are counted. To do so, we call `Keywords.make_count` on

16
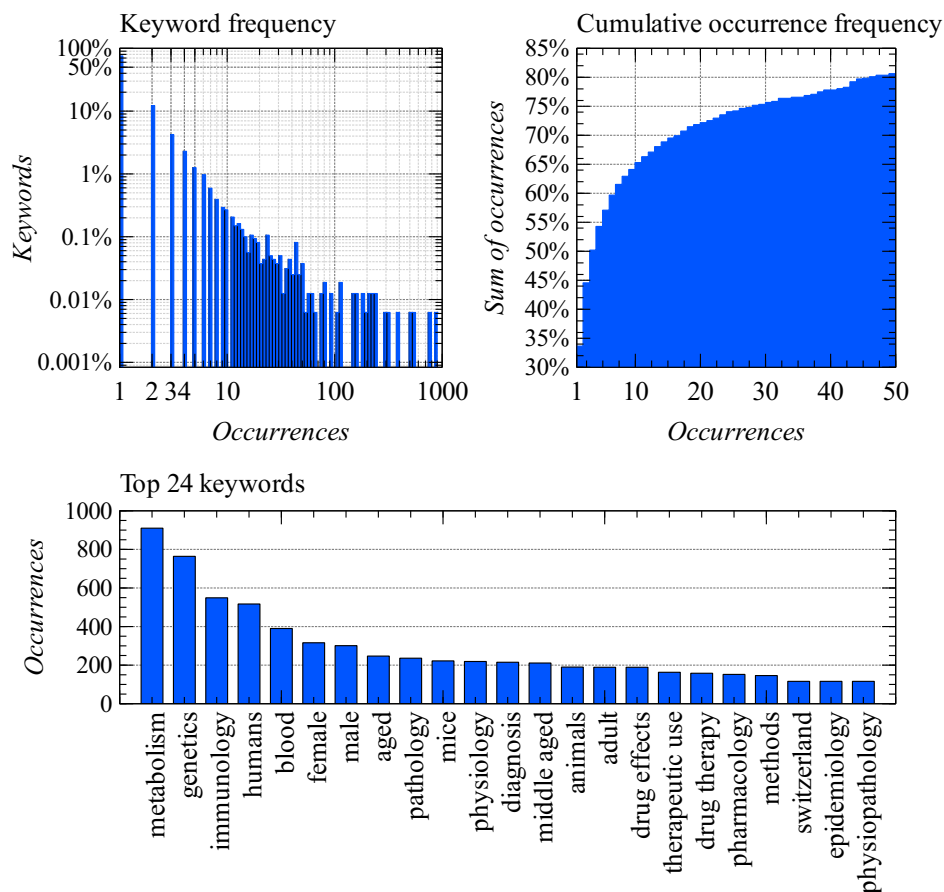
Figure 3: In `keywords/keywords_clean_histogram.json`, the distribution of keywords is strongly skewed right (min = Q1 = M = Q3 = 1, max = 910). However, even though keywords with only one occurrence constitute over 75% of the total keywords, their occurrences constitute less than 35% of the total occurrences. The most common keywords with 50 or more occurrences are extreme outliers but make up almost 20% of the total occurrences.

`/indexed/indexed_master_mesh_enriched.json` and save the output as
`/analysis/keywords_counted.json`:

```
sample_data_set = Utility.load_json(DIR + "/indexed/indexed_master.json")
keywords_counted = Keywords.make_count(sample_data_set)
Utility.save_json(keywords_counted, DIR + "/analysis/keywords_counted.json")
```

The median number of keywords for an item in the sample data set is 6 with
an IQR of 4 (min = 0, Q1 = 4, M = 6, Q3 = 8, max = 87). Of course, items
in the sample data set with a number of keywords below the first and above
the third quartile are highly suspect from a qualitative point of view: when
it comes to subject indexing, some terms are required, but more is usually
worse [!! source]. Items with too few or too many keywords will be discussed
in more detail in section section Assessment

5.4 Reconciliation

As explained in section X, Annif assigns index terms from a controlled
vocabulary. If we want to assess the quality of the indexing via a gold
standard, we must therefore ensure that the gold standard makes use of
the vocabulary used by Annif. As seen in section Y, the relevant (English)
vocabularies are Wikidata YSO. The next step in constructing the derivative
gold standard is hence to match the extracted and cleaned keywords with
keywords from YSO and Wikidata. This process is called "reconciliation"
(see https://docs.openrefine.org/manual/reconciling) and the tool of choice
for this task is OpenRefine (see section OpenRefine).

In this section, I will describe how the cleaned keywords were rec-
onciled with Wikidata and YSO, and which additional steps for
refinement were undertaken. In total, 2'104 data transformation oper-
ations were performed; the complete operation history is available as
`/keywords/operation_history.json`.

The data from `/keywords/keywords_clean_histogram.json` was imported
into OpenRefine. The `keyword` column was then duplicated and reconciled
with Wikidata. Here the parameters were chosen as follows: reconcile against
no particular type, and auto-match candidates with high confidence.

The reconciliation service API returns automatic matches (call them "sugges-
tions"). A suggestion is correct if and only if the meaning of the keyword from
`/keywords/keywords_clean_histogram.json` corresponds to the meaning
of the suggested concept from Wikidata. Note that for homonymous or

polysemous keywords, it is impossible to confirm a correct match without further context; those keywords therefore cannot be reconciled (but see section "Construction" for a possible solution). Unfortunately, random sampling showed that the overall quality of the reconciliation was not satisfactory, that is, there were too many incorrect suggestions. A two-pronged strategy was adopted to ameliorate the quality of the reconciliation.

First, the suggestions to the top 500 keywords were manually verified. These keywords account for 14'996 occurrences or 40.638% of the total occurrences and thus constitute an effective lever.

Second, systematic biases were identified and removed. The most prevalent bias was an due to a suggestion's type. As stated above, the reconciliation service API was not constrained by type but had access to the complete Wikidata database. Since Wikidata is an ontology that encompasses everything, it also features types whose concepts cannot qualify as subject terms (at least in the present context). The most prominent example is the type Q13442814 "scholarly article." Wikidata contains the metadata of many scholarly articles. Now, for some of our keywords, there is a scholarly article with a title that exactly matches the keyword; and since there is no restriction concerning the type, the scholarly article is suggested with high confidence (see https://github.com/OpenRefine/OpenRefine/wiki/Reconciliation-Service-API for details). For example, [keyword/article]. To generalize, suggestions with types whose concepts are proper names are usually incorrect. Based on this observation, suggestions with the following types were rejected: "scholarly article," "clinical trial," "scientific journal," "academic journal," "open access journal," "thesis," "doctoral thesis," "natural number." Suggestions with the following types were manually verified (i.e., checked for correctness): "human," "album," "film," "musical group," "business," "literary work," "television series," "organization," "family name," "written work," "video game," "single, television series episode," "painting," "commune of France," "city of the United States," "magazine," "studio album," "year," "nonprofit organization," "border town," "international organization," "political party," "software," "song," "website," "article comic strip," "collection," "commune of Italy," "fictional human," "film," "government agency," "village," "academic journal article," "female given name," "poem."

With these improvements in place, each keyword with a suggestion was assigned a QID via the "Add cloumns from reconciled values"-function (and similar for YSO and MeSH identifiers). The data was then exported and saved as /keywords/keywords_reference.json. Keywords with a QID

now constitute 69% of all keywords and 78% of their total occurrences in the sample data set, but these numbers are significantly lower for the MeSH identifier (53.6% of all keywords with only 28.8% of all occurrences) and especially low for the YSO identifier (26.9% of all keywords with 11% of all occurrences). In both cases, the problem is due to the fact that Wikidata's mapping of MeSH respectively YSO is only partial. There can be two reasons for this state of affairs for a given entry: either there is no match between Wikidata and YSO or MeSH (after all, Wikidata is much larger than MeSH and YSO taken together), or there is a match but it has not yet been added to the mapping. In the latter case, at least with respect to YSO, there is a solution: Finto provides a REST-style API to access the YSO vocabulary directly (see https://api.finto.fi/). For each keyword in the list of reference keywords that lacks a YSO identifier, the Finto API is queried; if a term turns up, it is added to the keyword. The effect of this second reconcilement is detailed in Figure 4. It is achieved by calling `Keywords.enrich_with_yso` on `/keywords/keywords_reference.json` and saving the output as `/keywords/keywords_reference_master.json`

```
keywords_reference = Utility.load_json(DIR + "/keywords/keywords_reference.json")
keywords_reference_new = Keywords.enrich_with_yso(keywords_reference)
Utility.save_json(keywords_histogram, DIR + "keywords/keywords_reference_master.json"
```

Finally, consider the distribution of the cleaned and reconciled keywords per item in the Edoc sample data set. The corresponding data is generated with `Keywords.make_count` as described in subsection Analysis above and available as `/analysis/keywords_counted.json`. Figure 5 shows that the median number of keywords from MeSH or YSO might be too low to be adequate. This problem can be amended by imposing further constraints on the sample data set and such a solution is discussed in section [!! section].

5.5 Discussion

Let us now turn to the evaluation of the reconciliation: how many of the suggestions were correct? This question was answered via random sampling (following Roth and Heidenreich 1993, 204–25). The random sample ($n = 500$) was created by calling `Analysis.make_random_sample` on `/keywords/reference_keywords_master.json`; it is available at `/analysis/random_keywords.json`. The sample was then imported into OpenRefine and the judgement (1 for correct, 0 for incorrect) of the manual verification was recorded in the column `verification`. The data was then exported and is available at `/analysis/random_keywords_verified.csv`.

20
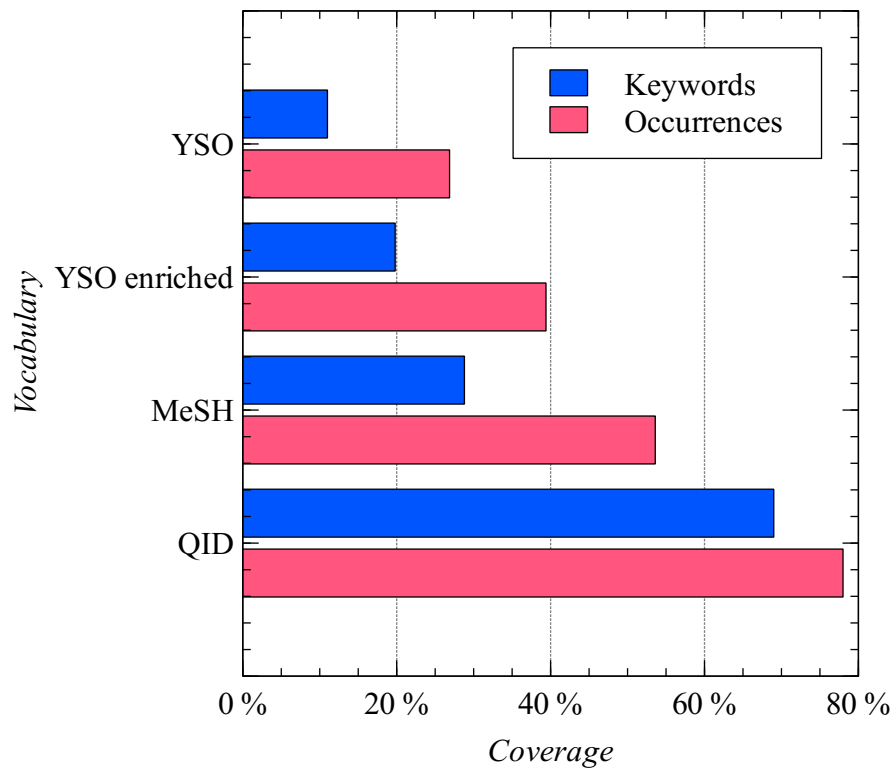
Figure 4: The coverage of `/keywords/keywords_reference_master.json` by the controlled vocabularies of Wikidata (QID), Medical Subject Headings (MeSH), and YSO (General Finnish Ontology). Both MeSH and YSO are dependent on QID but independent of each other. YSO enriched is the superset of YSO created by reconciling keywords that lack a YSO identifier directly with the YSO database provided by the Finto API; it is hence independent of Wikidata.
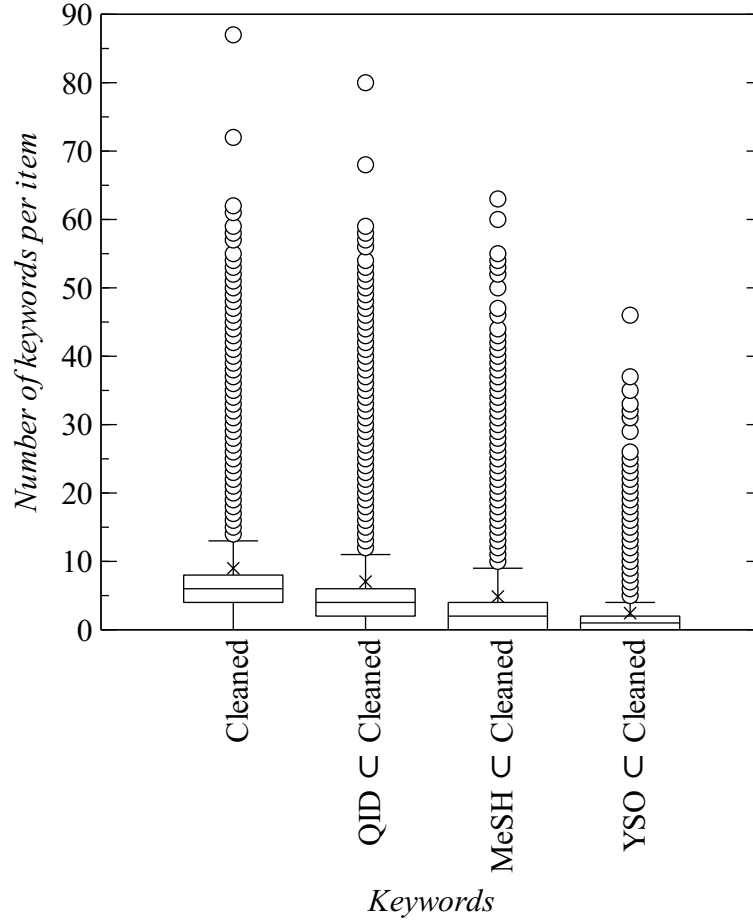
Figure 5: The distribution of keywords per item in the Edoc sample data set. The leftmost boxplot shows the distribution of cleaned keywords (min = 0, Q1 = 4, M = 6, Q3 = 8, max = 87); the other boxplots show the distribution of cleaned keywords with reconciled QID (min = 0, Q1 = 2, M = 4, Q3 = 6, max = 80), MeSH ID (min = Q1 = 0, M = 2, Q3 = 4, max = 63), and YSO ID respectively (min = Q1 = 0, M = 1, Q3 = 2, max = 46). All four distributions are similarly consistent, but there is a linear decrease of the distribution's center leaving MeSH and YSO with potentially too few descriptors to represent an adequate indexing.

An analysis of this data shows that 53% of the suggestions in the random sample were correct with a 95% confidence interval of 48.8% to 57.3%. We can therefore conclude that 8'507 ± 692.1 of the 16'050 keywords in `/keywords/reference_keywords.json` have correct suggestions. Note that of the 235 incorrect suggestions in the random sample, 188 were incorrect by default because they were missing a QID; the share of non-empty yet incorrect suggestions is only 9.4% in the random sample meaning that the quality of the reconciliation is not as disappointing as it might seem at first glance.

6 Assessment

In this section I will assess the quality of the sample data set's indexing with Annif based on the gold standard.

6.1 Precision, recall, F1-score

The metrics used for the assessment are precision, recall and F1-score. Precision and recall are standard metrics for indexing quality (e.g., Gantert 2016, 197) whereby the F1 score plays are more prominent role in the assessment of machine indexing (e.g., Suominen 2019, 11–14; Toepfer and Kempf 2016, 93f.). Of course, there is a host of alternative metrics (such as indexing consistency, transparency, reproducability) that are neglected here.

Let us briefly look at the definitions and motivations of the chosen metrics. Remember that a suggestion of a subject term is correct if and only if the subject term is in the derivative gold standard. The possible outcomes are summarized in Table 1.

		Suggested by Annif?	
		No	Yes
In gold standard?	No	True negative	False positive
	Yes	False negative	True positive

Table 1: Annif confusion matrix.

"Precision" is the fraction of the correctly suggested subject terms; a suggestion is correct if and only if it is in the derivative gold standard:

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

23

Or put as question: what fraction of the subject terms suggested by Annif are also in the derivative gold standard?

"Recall" is the fraction of correct subject terms out of all correct subject terms:

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}}$$

Put as question: what fraction of the subject terms in the gold standard were suggested by Annif?

The F1-score is the harmonic mean between precision and recall:

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

The above scoring metrics were implemented using the scikit-learn machine learning library (Pedregosa et al. 2011). For a given Annif configuration (section "Annif versus derivative gold standard")... [perhaps explain how it is done, important to mention transformation of multilabel to binary].

6.2 Creating the data foundation

I will now describe how the data foundation for the assessment was created. There are three parameters that need to be distinguished (see section "Annif"):

1. The Annif project (algorithm plus vocabulary) responsible for the indexing of the sample data set. As per the Annif REST API, these are `yso-en`, `yso-maui-en`, `yso-bonsai-en`, `yso-fasttext-en`, and `wikidata-en`.
2. The text base per item, namely title versus title and abstract.
3. The maximum number of suggestions per item. Since we required 10 suggestions per item, we can choose between 1-10 suggestions.

Combining these parameters, we really have 100 Annif configurations whose performance we want to compare and assess: Annif project $*$ text base $*$ maximum number of suggestions $= 5 * 2 * 10 = 100$. Remember the convention for constructing the marker that identifies a configuration (see section "Annif"): `{project_id}-{abstract}-{fulltext}-{n}-{threshold}`.

For each configuration, the F1-score was then computed. It is important to note that each metric comes in three different flavors dubbed "macro," "micro," and "weighted" respectively (see Sokolova and Lapalme 2009). In the

macro flavor, the metric represents simply the mean per class (i.e., correct or incorrect suggestion). The weighted metric is the macro metric but each class is weighted by its true positives. By contrast, a metric with the micro flavor is computed globally over all true positives and false positives respectively negatives. So the "macro" and "weighted" flavors are useful for assessing the performance of a configuration with respect to individual cases of assigning subject terms (call them "samples") whereas the "micro" flavor is most suitable to assess the overall performance of a configuration with respect to assigning subject terms.

To compute the F1-score, we call `Analysis.super_make_metrics` on `/indexed/indexed_master_mesh_enriched`. The 100 output files are saved as `/metrics/metrics_{marker}.json` where `marker` specifies the Annif configuration; a single file for analysis is saved as `/analysis/metrics.json`:

```
Analysis.super_make_metrics(DIR + "/indexed/indexed_master.json")
```

Note that the data foundation includes additional information, namely the sample size and the raw values for the confusion matrix. The sample size is a histogram of each instance in which a value in the confusion matrix was computed, that is, each case in which either Annif or the derivative gold standard assigned a subject term. Finally, note that if an item in the Edoc sample data set had an empty derivative gold standard, no score was computed; this is the case exactly if none of the cleaned keywords had been matched to Wikidata or YSO respectively. Configurations with a Wikidata vocabulary had a scoring coverage of 94.38% of the items in the sample data set as compared to only 62.84% for configurations with a YSO vocabulary.

6.3   General results

In this section I will discuss the general results of assessing the performance of the 100 Annif configurations versus the derivative gold standard with respect to the Edoc sample data set. The data foundation is available at `/analysis/metrics.json`.

Let us first consider overall performance. Here the most striking result is that Wikidata configurations outperformed YSO configurations (see Figure 6, left). However, the explanation of this effect is not evident. The two main explanatory hypotheses are 1. that the suggestions by the Wikidata configurations are more salient, or 2. that the derivative gold standard is biased towards Wikidata due to its higher coverage of QIDs as compared to YSO IDs. By contrast, the Wikidata configurations are more productive
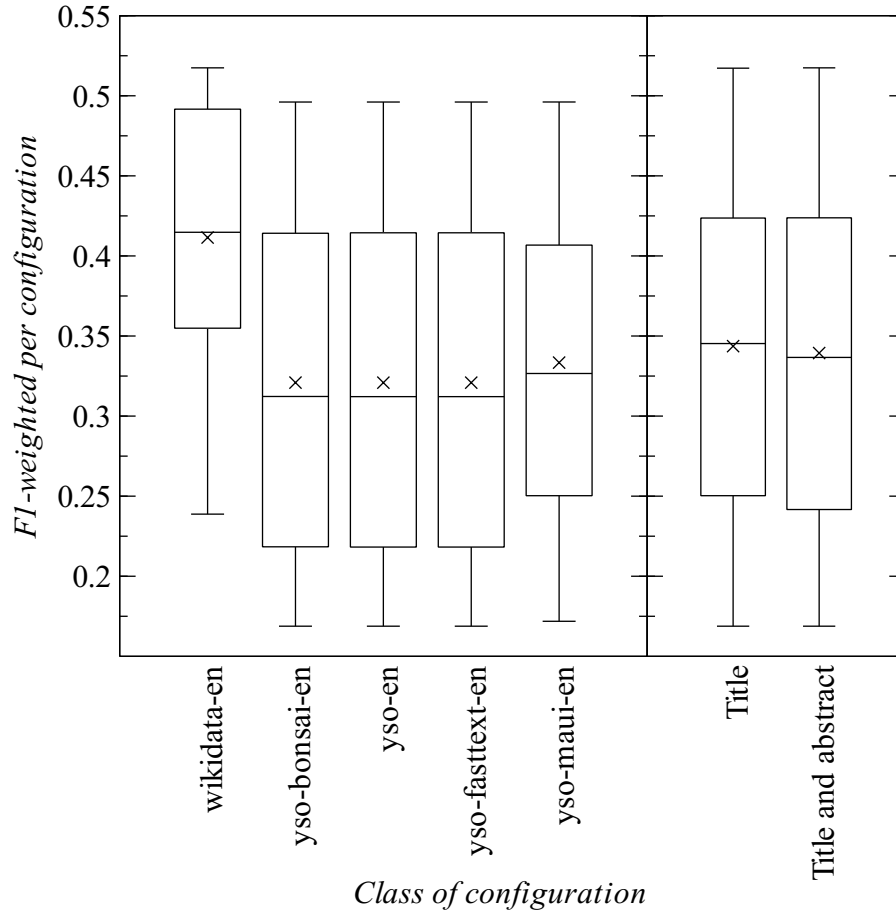
Figure 6: Distribution of weighted F1-scores per class of project of Annif configuration (left) and per class of text basis of Annif configuration (right). `wikidata` (min = 0.239, Q1 = 0.355, M = 0.415, Q3 = 0.492, max = 0.517) outperforms any YSO configuration. `yso-bonsai-en`, `yso-en` and `yso-fasttext-en` are on par (min = 0.169, Q1 = 0.218, M = 0.312, Q3 = 0.414, max = 0.496) and slightly outperformed by the more consistent `yso-maui-en` (min = 0.171, Q1 = 0.250, M = 0.327, Q3 = 0.407, max = 0.496). Surprisingly, the performance of configurations based on titles (min = 0.169, Q1 = 0.250, M = 0.345, Q3 = 0.424, max = 0.517) was marginally better than the performance of configurations based on titles and abstracts (min = 0.169, Q1 = 0.242, M = 0.337, Q3 = 0.424, max = 0.517).

	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
min	0.239	0.403	0.454	0.391	0.336	0.289	0.250	0.218	0.192	0.169
Q1	0.407	0.491	0.454	0.391	0.336	0.289	0.250	0.218	0.192	0.169
M	0.414	0.496	0.454	0.391	0.336	0.289	0.250	0.218	0.192	0.169
Q3	0.414	0.496	0.456	0.400	0.351	0.309	0.277	0.251	0.231	0.215
max	0.414	0.496	0.492	0.517	0.503	0.467	0.427	0.389	0.355	0.324

Table 2: Distribution of weighted F1-scores per maximum number of suggestions $1 \leq n \leq 10$.

than the YSO configurations, and the reason for this effect is the higher coverage of QIDs as compared to YSO IDs (see Figure 7). By "productivity" I mean that the absolute number of assigned subject terms. So Wikidata configurations are not only qualitatively but also quantitatively superior to YSO configurations.

Furthermore, the four YSO configurations are more or less on par: the `yso-bonsai-en`, `yso-en`, and `yso-fasttext-en` configurations perform very similarly and are slightly outperformed by `yso-maui-en` (see Figure 6, left). This result is consistent with other assessments reporting that "[o]f the individual algorithms, Maui performed best on all corpora" (Suominen 2019, 13).

A surprising result is that there was no performance gain achieved by increasing the text basis from titles to titles and abstracts (see section "Creating the data foundation": if anything, the performance for the extended text basis was slightly worse (see Figure 6, right). There is again no easy explanation for this find since there are multiple plausible causes which cannot be distangled here. For example, the abstract of an item in the Edoc sample data set might be less representative of the item's content than its title. In that case, the observed worse perfomance of the Annif configurations based on title and abstracts would even be expected. Alternatively, it might be the case that the problem lies with Annif: a larger text basis introduces more confounding factors and might make it harder to extract the subject terms required. An experimentum crucis to decide between the two discussed explanations would be to further extend the text basis to include full texts in addition to titles and abstracts.

Let us turn to the parameter of the maximum number of suggestions per item in the Edoc sample data set (see Figure 8). Here the best performance was achieved by a token $n = 4$ configuration. However, the type $n = 4$
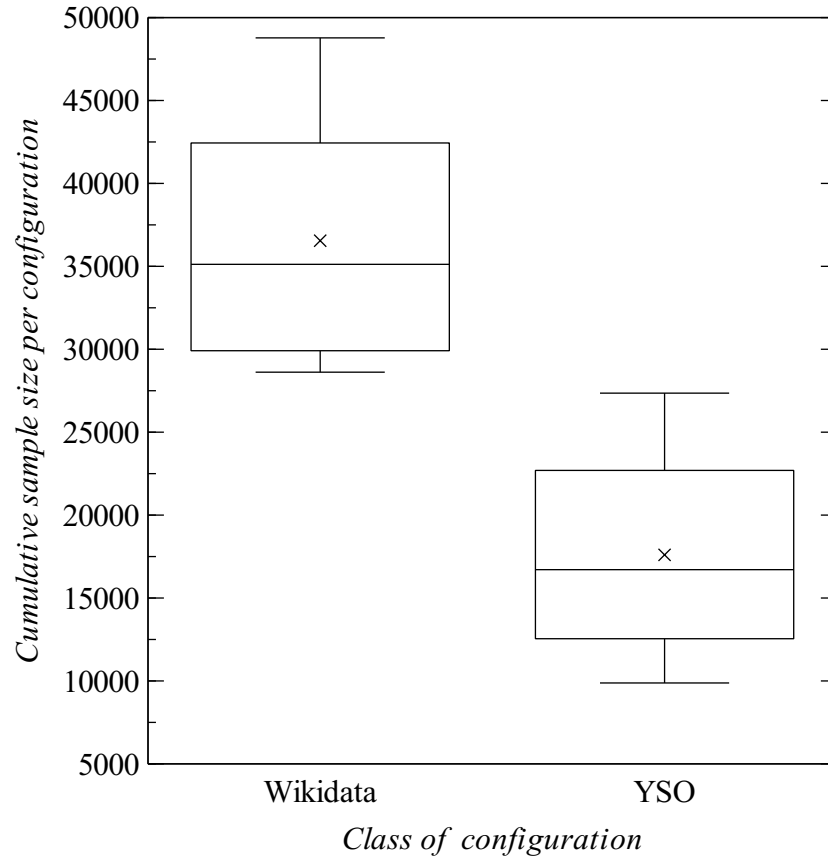
Figure 7: The distribution of cumulative sample size per project class of Annif configuration. Wikidata configurations (min = 28'618, Q1 = 29'912, M = 35'121, Q3 = 42'439, max = 48'776) are more productive than YSO configurations (min = 9'878, Q1 = 12'546, M = 16'710, Q3 = 22'694, max = 27'354).

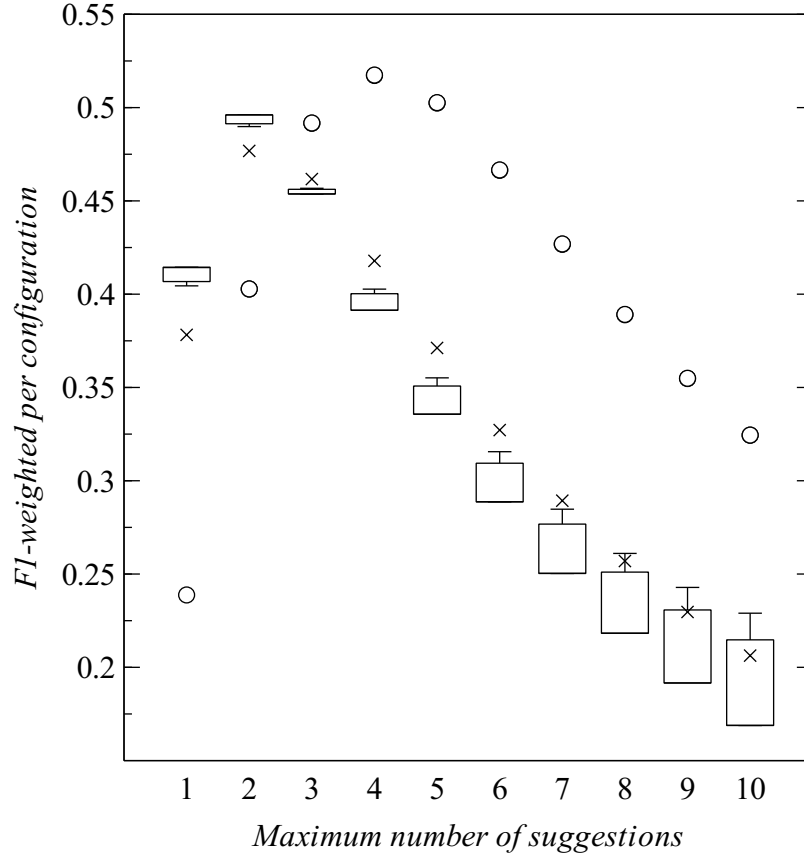Figure 8: Distribution of weighted F1-scores per maximum number of suggestions $1 \leq n \leq 10$. See Table 2 for numeric values. $n = 2$ is the best perfomring type of configuration. However, it is outperformed by $n = 4$ and $n = 5$ token configurations. The distribution of the distributions is skewed left. Types of configurations with a smaller maximum number of suggestions display a more consistent performance.

configuration performed on average significantly worse than the type n = 3 and type n = 2 configurations. So with respect to the maximum number of suggestions, there is a discrepancy between the best performing token configuration and the best performing type of configuration. This distinction is important: when choosing a configuration for the purpose of a production system, we are usually interested in the best token configuration. The top 20 token configurations are summarized in Figure 9. The best performing configuration with a weighted F1-score of 0.517 is `wikidata-en` with a maximum number of 4 suggestions per item; the text basis (title or abstract and title) does not matter.
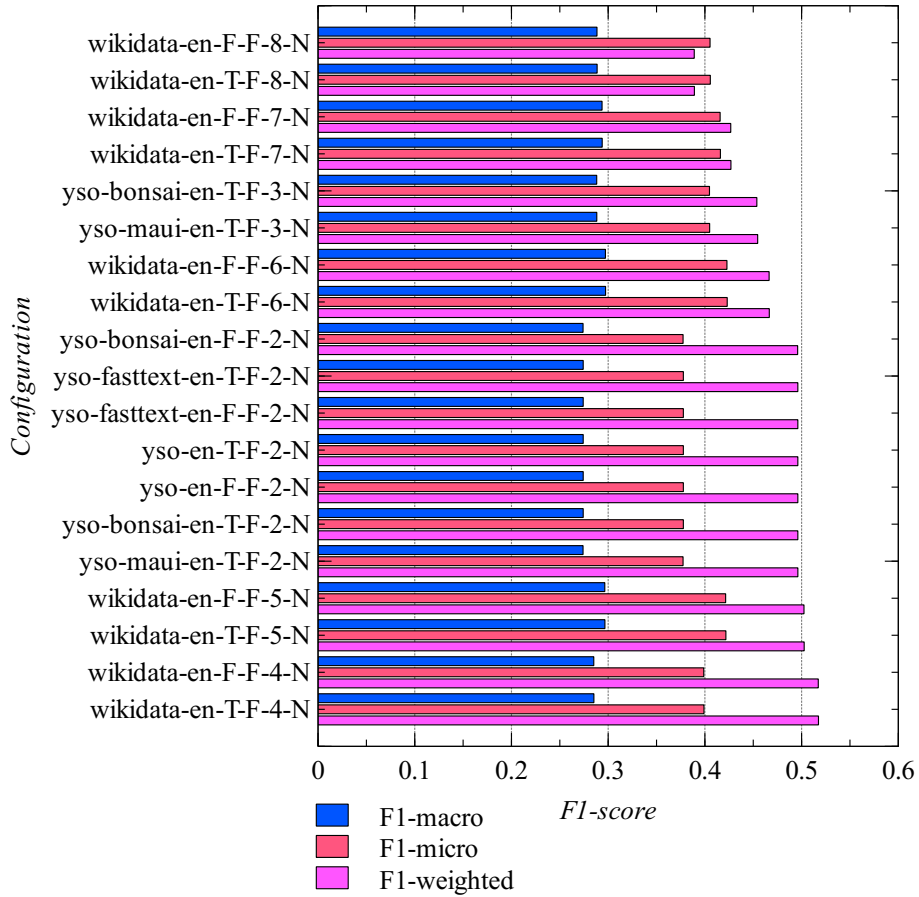

Figure 9: 20 best performing Annif configurations (according to F1-score).

30

6.4 Departmental results

In section "General results" I have identified and discussed the overall best Annif configrations. However, in section "Analysis") I had also noted the caveat that the performance of Annif might vary according to department due to systematic biases in constructing the Edoc sample data set. In this section I will therefore assess the performance of the various Annif configurations per department.

We begin again by constructing the data foundation. Since the Edoc `department` data field is mandatory, the sample data set is partitioned into blocks of departments by default. We can thus create a data foundation for each such block by calling `Analysis.super_make_metrics` with a departmental parameter:

```
for dept in Data.get_departments():
    Analysis.super_make_metrics(DIR + "/indexed/indexed_master.json", dept)
```

This yields 1000 files in `/metrics/metrics_{department}_{marker}.json` (100 configurations ∗ 10 departments) where `department` specifies the department according to the Edoc convention. The data foundation is summarized in `/analysis/metrics.json`.

Let us now look at the results. First consider the distribution of the performance scores across departments as shown in Figure 11. We can see that there are striking differences in variability between departments. The departments with the most consistent weighted F1-scores are Medicine and Central Services. However, these two departments have also the lowest maximum weighted F1-scores. It is noteworthy that all other departments have maximum weighted F1-scores that are well above the weighted F1-score of 0.517 of the best performing general Annif configuration.

Let us now consider the best performing Annif configurations per department as summarized in Figure 12. It is evident that only two of the ten departments (namely Associated and Central Services) have as best performing configuration the configuration that was declared the overall best performing configuration (namely `wikidata-en-F-F-4-N`). More surprisingly, YSO outperforms Wikidata in all other departments except Interdisciplinary. However, the margin is rather slim. The notable exception is the department Economics, where the YSO configuration outperforms the Wikidata configuration by a factor of 3. More importantly, the best performing YSO configurations are significantly less productive than the slightly worse per-
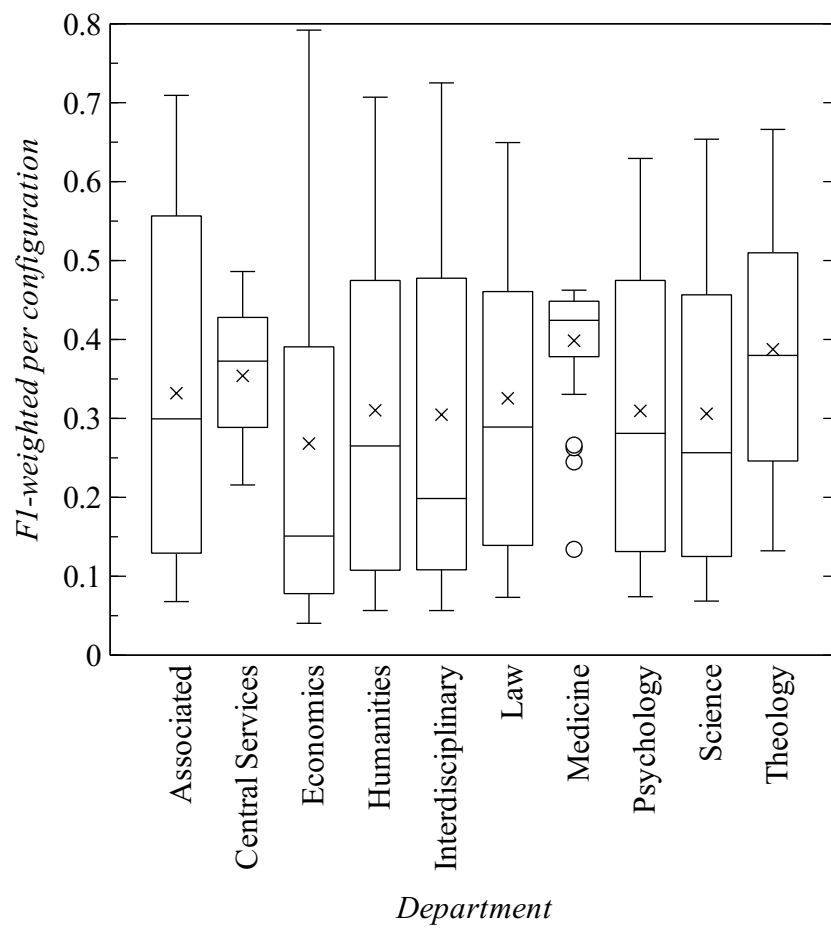
Figure 10: Distribution of weighted F1-scores for all Annif configurations per department.
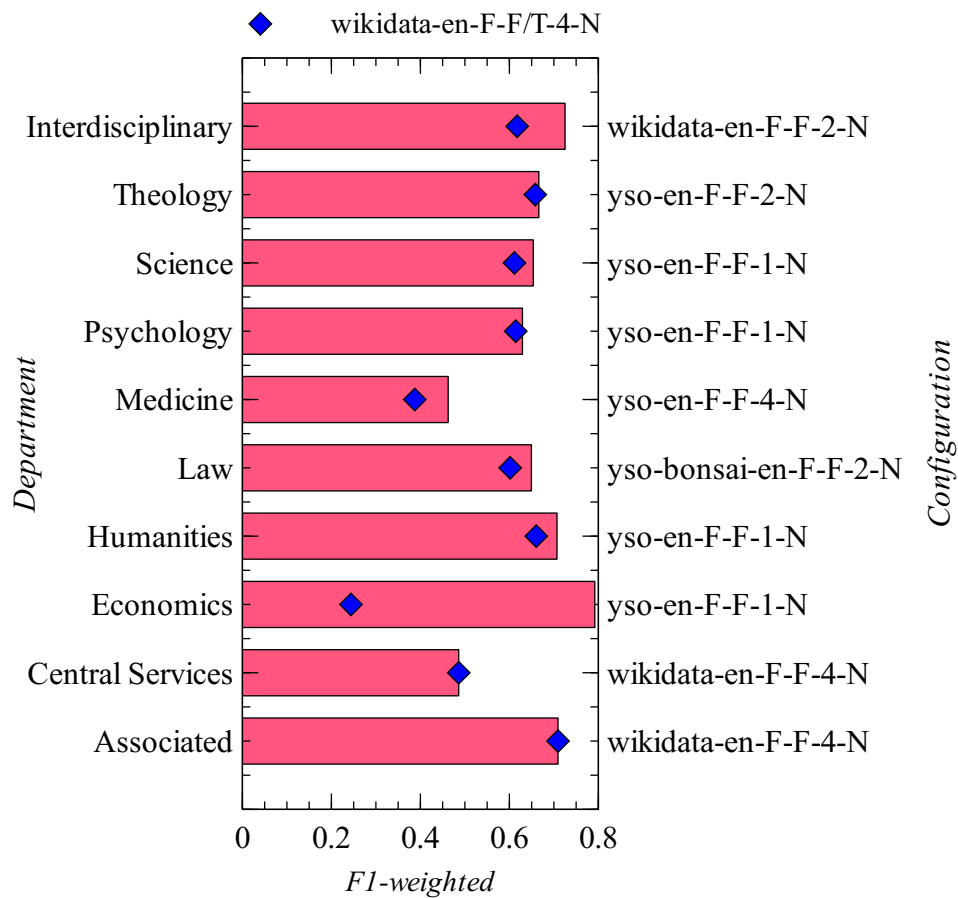
Figure 11: Best performing Annif configuration (according to weighted F1-score) per department as compared to the overall best performing Annif configuration `wikidata-en-F-F-4-N`.

forming Wikidata configuration, except for `yso-en-F-F-4-N` in the Medicine department.

7   Outlook and conclusion

7.1   Refinement

- fulltext support
- mesh; A gold standard is usually the product of manual indexing by "information experts, subject experts, or potential or real end users" (Golub et al. 2016, 10). This entails its own host of epistemic problems relating to objectivity and consistency of the assigned subject terms. Most importantly, however, the construction of a gold standard from scratch is very expensive. It is therefore not an option for the project at hand. Rather, I will construct what could be called derivative gold standards by reusing indexing data that is already available. Here we can distinguish two distinct kinds of derivative gold standards: first, I will construct a derivative gold standard based on the author keywords available in the sample data set; call this the "derivative" gold standard. Second, I will construct a derivative gold standard based on indexing metadata available in repositories distinct from Edoc; call this the "foreign" gold standard. -We enrich the sample items with MeSH keywords from PubMed if available (item needs a PubMed ID and items needs to be indexed with MeSH on PubMed, 1653 items match this requirement); the resulting file is indexed_master_mesh.json. Like so: Data.enrich_with_mesh(DIR + "/indexed/indexed_master.json," DIR + "/indexed/indexed_master_mesh")

7.2   Implementation strategy

8   Appendix

Codebase available at: https://github.com/MHindermann/mas

8.1   class files.Utility()

A collection of utility functions.

8.1.1   classmethod load_json(file_path)

Load a JSON object from file.

- **Parameters**

  file_path (`str`) – complete path to file including filename and extension

- **Return type**

  `list`

8.1.2 classmethod save_json(data, file_path)

Save data as JSON file.

- **Parameters**

  - **data** (`Union[List, Dict]`) – the data to be saved
  - **file_path** (`str`) – complete path to file including filename and extension

- **Return type**

  `None`

8.1.3 classmethod split_json(file_path, save_path)

Split a JSON file into files not larger than 100MB.

- **Parameters**

  - **file_path** (`str`) – complete path to file including filename and extension
  - **save_path** (`str`) – complete path to save folder including filename without extension

- **Return type**

  `None`

8.2 class files.Data()

A collection of Edoc data functions.

8.2.1   classmethod enrich_author_keywords(file_path, save_path)

Enrich author keywords.

For each Edoc item: the string of author keywords is cut into single keywords and each keyword is cleaned. Each keyword is then enriched with Qid, MeSH and YSO ID if available.

- **Parameters**

    - **file_path** (`str`) – complete path to file including filename and extension

    - **save_path** (`str`) – complete path to save folder including filename without extension

- **Return type**

    `None`

8.2.2   classmethod enrich_with_annif(file_path, save_path, project_ids, abstract=False, fulltext=False, limit=None, threshold=None)

Enrich items from file with automatic keywords using Annif-client.

Available Annif-client project IDs are yso-en, yso-maui-en, yso-bonsai-en, yso-fasttext-en, wikidata-en.

- **Parameters**

    - **file_path** (`str`) – complete path to file including filename and extension

    - **save_path** (`str`) – complete path to save folder including filename without extension

    - **project_ids** (`List[str]`) – Annif-client project IDs to be used

    - **abstract** (`bool`) – toggle use abstract for indexing, defaults to False

    - **fulltext** (`bool`) – toggle use fulltext for indexing, defaults to False

    - **limit** (`Optional[int]`) – Annif-client limit, defaults to None

    - **threshold** (`Optional[int]`) – Annif-client threshold, defaults to None

36

- **Return type**

  None

8.2.3 classmethod enrich_with_mesh(file_path, save_path)

Enrich Edoc data per item with MeSH keywords from PubMed if available.

- **Parameters**

  – **file_path** (`str`) – complete path to file including filename and extension

  – **save_path** (`str`) – complete path to save folder including filename without extension

- **Return type**

  None

8.2.4 classmethod fetch_mesh(pubmed_id)

Fetch MeSH keywords for article based on PubMed ID.

- **Parameters**

  pubmed_id (`str`) – article PubMed ID

- **Return type**

  List[Dict]

8.2.5 classmethod get_departments()

Get all Edoc departments.

- **Return type**

  List[str]

8.2.6 classmethod inspect(data, *fields)

Print fields of items in data.

- **Parameters**

  – **data** (`List[Dict]`) – data to be inspected

  – **fields** (`str`) – fields to focus on

- **Return type**

  ```
  None
  ```

8.2.7 classmethod map2reference(keyword)

Map a keyword to its reference keyword.

Keyword must first be cleaned by corresponding _Keyword method.

- **Parameters**

  keyword (`str`) – the keyword

- **Return type**

  ```
  Dict
  ```

8.2.8 classmethod select_from_data(data, *fields)

Select items from data based on non-empty fields.

Only items where all required fields are non-empty are sample.

- **Parameters**

  - **data** (`List[Dict]`) – the input data
  - **fields** (`str`) – the required fields for an item to be sample

- **Return type**

  `List[Dict]`

8.2.9 classmethod select_from_file(file_path, *fields)

Select items from file according to fields.

For example: select_from_file(DIR + "/raw/2019.json," "title," "abstract," "keywords," "id_number")

- **Parameters**

  - **file_path** (`str`) – complete path to file including filename and extension
  - **fields** (`str`) – the required fields for an item to be sample

- **Return type**

  None

8.2.10   classmethod super_enrich_with_annif(abstract)

Enrich items with automatic keywords using all Annif-client projects.

- **Parameters**

  abstract (`bool`) – toggle use abstract for indexing

- **Return type**

  None

8.3   class files.Keywords()

A collection of functions for manipulating Edoc author keywords.

8.3.1   classmethod clean_keyword(keyword)

Clean a keyword.

- **Parameters**

  keyword (`str`) – the keyword

- **Return type**

  List[str]

8.3.2   classmethod clean_keywords(keywords_per_item)

Turn a string of keywords per item into a list of cleaned (=normalized) keywords.

- **Parameters**

  keywords_per_item (List[str]) – the keywords

- **Return type**

  List[str]

8.3.3 classmethod enrich_with_yso(file_path, save_path)

Blah.

- **Parameters**

    - **file_path** (`str`) – complete path to file including filename and extension

    - **save_path** (`str`) – complete path to save folder including filename without extension

8.3.4 classmethod extract_keywords(file_path)

Extract keywords per item from file.

- **Parameters**

    file_path (`str`) – complete path to file including filename and extension

- **Return type**

    `List[str]`

8.4 classmethod fetch_yso(keyword)

Fetch the YSO ID for a keyword if any.

- **Parameters**

    keyword (`str`) – the keyword

- **Return type**

    `Optional[str]`

8.4.1 classmethod make_count(file_path, save_path)

Count all keywords and their respective IDs per item in file.

- **Parameters**

    - **file_path** (`str`) – complete path to file including filename and extension

    - **save_path** (`str`) – complete path to save folder including filename and extension

- **Return type**

  `None`

8.4.2   classmethod make_histogram(keywords)

Make a histogram for keywords.

- **Parameters**

  keywords (`List[str]`) – the keywords

- **Return type**

  `List[Dict]`

8.5   class files.Analysis()

A collection of data analysis functions.

8.5.1   classmethod count_confusion(standard, suggestions)

Count the values in the confusion matrix for standard and suggestions.

- **Parameters**
  - **standard** (`list`) – the standard
  - **suggestions** (`list`) – the suggestions

8.5.2   classmethod extract_standard(item, marker)

Extract gold standard IDs from item.

- **Parameters**
  - **item** (`dict`) – the Edoc item
  - **marker** (`str`) – the type of ID

- **Return type**

  `Optional[list]`

8.5.3 classmethod extract_suggestions(item, marker, n=10)

Extract top n Annif suggestion IDs from item.

- **Parameters**
    - **item** (`dict`) – the Edoc item
    - **marker** (`str`) – project_id-abstract-fulltext-limit-threshold
    - **n** (`int`) – number of top IDs (by score) to be extracted, defaults to 10

8.5.4 classmethod get_id_type(marker)

Get the ID type from the project ID.

- **Parameters**

    marker (`str`) – project_id-abstract-fulltext-limit-threshold

- **Return type**

    `str`

8.5.5 classmethod get_sklearn_array(item, project_id, abstract=False, fulltext=False, limit=None, threshold=None, n=10)

Get Sklearn y_true and y_pred for an item.

Available Annif-client project IDs are yso-en, yso-maui-en, yso-bonsai-en, yso-fasttext-en, wikidata-en.

- **Parameters**
    - **item** (`dict`) – sample data set item
    - **project_id** (`str`) – Annif-client project ID
    - **abstract** (`bool`) – toggle use abstract for indexing, defaults to False
    - **fulltext** (`bool`) – toggle use fulltext for indexing, defaults to False
    - **limit** (`Optional[int]`) – Annif-client limit, defaults to None
    - **threshold** (`Optional[int]`) – Annif-client threshold, defaults to None

- **n** (`int`) – number of top IDs (by score) to be extracted, defaults to 10

- **Return type**

  Optional[dict]

8.5.6 classmethod make_metrics(file_path, project_id, abstract=False, fulltext=False, limit=None, threshold=None, n=10, department=None)

Make Sklearn metrics F1, recall, precision for file.

The output is saved as /metrics/metrics_{marker}.json.

Available Annif-client project IDs are yso-en, yso-maui-en, yso-bonsai-en, yso-fasttext-en, wikidata-en.

- **Parameters**

  - **file_path** (`str`) – complete path to file including filename and extension

  - **project_id** (`str`) – Annif-client project ID

  - **abstract** (`bool`) – toggle use abstract for indexing, defaults to False

  - **fulltext** (`bool`) – toggle use fulltext for indexing, defaults to False

  - **limit** (Optional[`int`]) – Annif-client limit, defaults to None

  - **threshold** (Optional[`int`]) – Annif-client threshold, defaults to None

  - **n** (`int`) – number of top IDs (by score) to be extracted per item, defaults to 10

  - **department** (Optional[`str`]) – restrict to items from department

- **Return type**

  None

8.5.7 classmethod make_random_sample(file_path, save_path, size)

Save a random sample from the population in the file.

- **Parameters**
  - **file_path** (`str`) – complete path to file including filename and extension
  - **save_path** (`str`) – complete path to save folder including filename and extension
  - **size** (`int`) – the sample size
- **Return type**

  None

8.5.8   classmethod print_chi_square_fit(file_path)

Print chi square goodness of fit.

File must be CSV with first line title; three columns with first column categories, second column expected and third column observed values.

- **Parameters**

  file_path (`str`) – complete path to file including filename and extension
- **Return type**

  None

8.5.9   classmethod super_make_metrics(file_path, department=None)

Make metrics for all combinations of Annif projects and parameters in enriched Edoc file.

Output files are saved in /metrics/. Joint results are saved in /analysis/metrics.json.

- **Parameters**
  - **file_path** (`str`) – complete path to file including filename and extension
  - **department** (`Optional[str]`) – restrict to items from department

8.5.10 classmethod super_make_stats()

Make metrics for files in /metrics.

Output is saved as /analysis/metrics.json.

- **Return type**

  None

9 Bibliography

Gantert, Klaus. 2016. *Bibliothekarisches Grundwissen.* 9., vollständig neu bearbeitete und erweiterte Auflage. Berlin: De Gruyter Saur. https://doi.org/10.1515/9783110321500.

Golub, Koraljka. 2019. "Automatic Subject Indexing of Text." *Knowledge Organization* 46 (2): 104–21. https://doi.org/10.5771/0943-7444-2019-2-104.

Golub, Koraljka, Dagobert Soergel, George Buchanan, Douglas Tudhope, Marianne Lykke, and Debra Hiom. 2016. "A Framework for Evaluating Automatic Indexing or Classification in the Context of Retrieval." *Journal of the Association for Information Science and Technology* 67 (1): 3–16. https://doi.org/10.1002/asi.23600.

Louis, Annie, and Ani Nenkova. 2013. "Automatically Assessing Machine Summary Content Without a Gold Standard." *Computational Linguistics* 39 (2): 267–300. https://doi.org/10.1162/COLI%7B/textunderscore%20%7Da%7B/textunderscore%20%7D00123.

Nagelschmidt, Matthias. 12.11.2020. "Evaluation von Annif für Die Maschinelle Inhaltserschließung an Der Deutschen Nationalbibliothek." https://wiki.dnb.de/download/attachments/181735291/07-Evaluation VonAnnif.pdf?version=1&modificationDate=1605778295000&api=v2.

Parke, Carol. 2013. *Essential First Steps to Data Analysis: Scenario-Based Examples Using SPSS.* Thousand Oaks: SAGE Publications. https://doi.org/10.4135/9781506335148.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research*, no. 12: 2825–30. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

Roth, Erwin, and Klaus Heidenreich. 1993. *Sozialwissenschaftliche Methoden: Lehr- Und Handbuch für Forschung Und Praxis.* 3., völlig überarbeitete und erweiterte Auflage. München: R. Oldenbourg Verlag.

Sokolova, Marina, and Guy Lapalme. 2009. "A Systematic Analysis of Performance Measures for Classification Tasks." *Information Processing & Management* 45 (4): 427–37. https://doi.org/10.1016/j.ipm.2009.03.002.

Stokhof, Martin, and Michiel van Lambalgen. 2011. "Abstractions and Idealisations: The Construction of Modern Linguistics." *Theoretical Linguistics* 37 (1-2): 1–26. https://doi.org/10.1515/thli.2011.001.

Suominen, Osma. 2019. "Annif: DIY Automated Subject Indexing Using Multiple Algorithms." *LIBER Quarterly* 29 (1): 1–25. https://doi.org/10.18352/lq.10285.

Suominen, Osma, Juho Inkinen, Tuomo Virolainen, Moritz Fürneisen, Bruno P. Kinoshita, Sara Veldhoen, Mats Sjöberg, Philipp Zumstein, Robin Neatherway, and monalehtinen. 2021. "NatLibFi/Annif: Annif 0.51." https://doi.org/10.5281/zenodo.4521923.

Toepfer, Martin, and Andreas Oskar Kempf. 2016. "Automatische Indexierung Auf Basis von Titeln Und Autoren-Keywords – Ein Werkstattbericht." *027.7 Zeitschrift für Bibliothekskultur / Journal for Library Culture* 4 (2): 84–97. https://0277.ch/ojs/index.php/cdrs_0277/article/view/156/354.

Universität Basel. n.d. "Research Database of the University of Basel User Manual." Edited by Universität Basel.