Problem

• How to semi-automate subject indexing with very little data?

• The title and author of a book is not typically enough information for a robust subject indexing process.

• We use metasearch (multiple search targets) to gather probable background information to inform the process.
Semi-automated processes were explored to support and extend, not replace, professional expertise.
What tools exist for automating subject indexing?

- **Annif**: An open-source tool from National Library of Finland has multiple algorithms to use and an open framework for training in a given vocabulary.

https://annif.org/
Backend: fastText

Juho Inkinen edited this page on Sep 23, 2022 · 12 revisions

The `fasttext` backend implements a text classification algorithm based on word embeddings and machine learning. It is a wrapper around the fastText library created by Facebook Research. The model resembles a feed-forward neural network with one hidden layer, though some shortcuts are used for computational efficiency.

The quality of results can be very good, but many parameters have to be selected to get optimal results. Training can be computationally intensive; by default it can train using all cores in parallel. If you have a machine with a huge number of CPU cores (more than 8), it is probably wise to limit the number of cores used for training; a good starting point is `thread=12` beyond which additional CPU cores do not significantly speed up the training, so you will just end up wasting CPU resources.

Installation

See Optional features and dependencies
Backend: Omikuji

Juho Inkinen edited this page on Jan 23, 2023 · 10 revisions

The `omikuji` backend is a wrapper around Omikuji, an efficient implementation of a family of tree-based machine learning algorithms. It can emulate Parabel and Bonsai, two state-of-the-art algorithms for extreme multilabel classification.

The quality of results has generally been extremely good, even without tuning of hyperparameters. Training can be computationally intensive; by default it will use all available CPU cores in parallel during the training phase. Also large amounts of RAM (several GB) may be required during training, but during use the memory usage is lower.

See also the `Annif-tutorial exercise about Omikuji project`.

Installation

See `Optional features and dependencies`
We ensemble the algorithms

Backend: nn_ensemble
Juho Inkinen edited this page on Jan 23, 2023 · 10 revisions

The `nn_ensemble` backend implements a trainable dynamic ensemble that intelligently combines results from multiple projects. Subject suggestion requests to the ensemble backend will be re-routed to the source projects. The results from the source projects will be re-weighted using a neural network implemented using Keras and TensorFlow 2. Initially (with no training), the backend behaves exactly like a basic `ensemble` backend, but the training data is then used to fine-tune the output to better match the given manually assigned subjects in the training documents.

This backend is similar in spirit to the `PAV` backend, but unlike PAV, this backend also supports online learning, which means that it can be further trained during use via the `learn` command in the Annif CLI and REST API.

TensorFlow may be able to utilize a GPU if the system has one available – see the TensorFlow documentation for details. In practice, when training an ensemble, the great majority of processing time is typically spent processing the document in the source projects, so the extra benefit of a GPU is likely to be minimal.

See also the Annif-tutorial exercise about NN ensemble project.

Installation

See Optional features and dependencies
Web search + Annif + LLM (llamafiler) filtering

- **Search** title/author data in metasearch system
- **Filter** search results to only those that match the book title/author.
- **Identify** entities with GLiNER and a schema of types that map to the facets of the target subject vocabulary.
- **Send only the entities** of web searches to our nn Annif API, which is trained on many examples from Penn records and others.
- **Filter** Annif suggestions with local LLM (LLama3)
- **Validate** a representative sample of LLM filtered suggestions with human specialists.
Web Search Tool

https://github.com/searxng/searxng
Search Engine Targets

<table>
<thead>
<tr>
<th>Search Engine</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brave</td>
<td><a href="https://search.brave.com/">https://search.brave.com/</a></td>
</tr>
<tr>
<td>DuckDuckGo</td>
<td><a href="https://lite.duckduckgo.com/lite/">https://lite.duckduckgo.com/lite/</a></td>
</tr>
<tr>
<td>Google</td>
<td><a href="http://www.google.com/search">www.google.com/search</a></td>
</tr>
<tr>
<td>QWant</td>
<td><a href="https://www.qwant.com/">https://www.qwant.com/</a></td>
</tr>
<tr>
<td>Alternative Front-end for Google Translate</td>
<td><a href="https://lingva.ml/">https://lingva.ml/</a></td>
</tr>
<tr>
<td>Dictionaries</td>
<td><a href="https://dictzone.com/dictionaries/">https://dictzone.com/dictionaries/</a></td>
</tr>
<tr>
<td>Wikidata</td>
<td><a href="https://www.wikidata.org/">https://www.wikidata.org/</a></td>
</tr>
<tr>
<td>Wikipedia</td>
<td><a href="https://www.wikipedia.org/">https://www.wikipedia.org/</a></td>
</tr>
</tbody>
</table>
Un análisis detallado de las decisiones históricas que resultaron en la actual ruta del ferrocarril al Atlántico. El autor expone las influencias que guidaron el desarrollo del transporte en esta región.

Let us know. If not, help out and invite Rafael to Goodreads. Rafael Ángel Jiménez Fajardo is the author of Viejos Cuentos de Viejos (3.90 average rating).

Title: Las ""rocas de Fajardo"" Y El Ferrocarril Por Tucurrique · Author: Rafael Ángel Jiménez Fajardo · Publisher: Independently Published ·


Buy Las ""rocas de Fajardo"" Y El Ferrocarril Por Tucurrique by Rafael Ángel Jiménez Fajardo online at Alibris. We have new and used copies available, in 1 editions.
Web searches are filtered for relevance

• If the tokens from the title/author aren’t present in the search result, the search result is not used in the follow-on steps.
• The **filtering** here uses a **fuzzy match of tokens** from the title/author to the search result.
Identify entities (NER)

• This is a named entity recognition (NER) task.
• We can leverage what we know about our target vocabulary and to direct the entity outputs toward our schema of types.
• The FAST vocabulary facets:
  **Names** (person and corporate), **Event**, **Uniform Titles**, **Chronological**, **Topical**, **Geographic**, **FormGenre**, **Meeting**.*

* information from [FAST (Faceted Application of Subject Terminology) Data](https://www.oclc.org) is made available by OCLC Online Computer Library Center, Inc. under the [ODC Attribution License](https://creativecommons.org/licenses/by/4.0/).
Direct **NER outputs to** FAST schema types

<table>
<thead>
<tr>
<th>FAST Facet</th>
<th>GLiNER Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names (Personal and Corporate)</td>
<td>Person, family, organization</td>
</tr>
<tr>
<td>Event</td>
<td>Event</td>
</tr>
<tr>
<td>Uniform Title</td>
<td>Title</td>
</tr>
<tr>
<td>Chronological</td>
<td>Time period</td>
</tr>
<tr>
<td>Topical</td>
<td>Concept, object</td>
</tr>
<tr>
<td>Geographic</td>
<td>Place</td>
</tr>
<tr>
<td>FormGenre</td>
<td>Genre</td>
</tr>
<tr>
<td>Meeting</td>
<td>Meeting</td>
</tr>
</tbody>
</table>
Use Annif to generate FAST subjects from identified entities

- We are sending the entities identified from web searches to our local Annif API.
LLM Quality Assurance

Once Annif returns labels from the FAST vocabulary, a final step of using the Llama3 llamfile will check to see if the subjects are relevant to the web search.

These are candidate subjects.
All LLM tasks run locally with Llamafile(s)

Simple grammar to generate yes/no response

• “LLM output is normally wild and unpredictable.

• You can control it is by imposing language constraints using Backus-Naur Form.

• This works by restricting which tokens can be selected when generating the output.

• For example, the flag --grammar 'root ::= "yes" | "no"' will force the LLM to only print "yes\n" or "no\n" and then exit().”

Set temperature to zero for a reproduceable result

```bash
# Execute the LLAMA3 command with improved grammar and error handling.
result=$("$LLAMA3"
    --temp 0 -ngl 35
    --grammar 'root ::= "yes" | "no"'
    -p "[INST]Is the label '$concept_label' a good representation of the main ideas in this text: '$search_text'?[/INST]"
    --silent-prompt 2>&1)
```
Human Quality Assurance

• A representative sample of the topics that an LLM has checked and have a human evaluator go over the suggestions.
Sample subjects

MMSID,Summary,Label

9978142013103680.0,"titled "Las 'Rocas de Fajardo' el Ferrocarril por Tucurrique"", analyzes historical decisions led current route railroad Atlantic. author exposes political economic influences often prevailed technical criteria explains town Tucurrique lost struggle influence initiated families vying power. also offers insights impact railroad's construction local population, economy, environment." "['Railroads--Design and construction', 'Railroad engineering', 'Cities and towns']"
Using Microsoft Power Automate, a library staff member could upload an excel file with the necessary schema to the Annif OneDrive folder.

- **Schema:** mmsid, titleauthor
- **Note:** Titleauthor column may include any relevant text to search the web, may include ISBN, ISSN, summary.

The Power Automate Flow Detects when a new file is in the OneDrive folder

- Checks once a minute
- If a new file is found, it sends it to an Airflow server for the multi-step processing.
• Power Automate Flow
Apache Airflow Process

- Apache Airflow is used for automating processes. The directed acyclic graphs (DAG) are written in python, and each of the tasks is a python file. The individual task can communicate with an Airflow feature **Xcom**, which sends the output file of the previous process to input into the next step.

- **Power automate sends the new files from OneDrive into Airflow.**
  - checks to see if it has seen this file previously. If the filename is new to Annif, it will run the whole process. If it has already run the process on this file, it will not run it again.
from datetime import datetime
from airflow import DAG
from airflow.operators.python import PythonOperator, ShortCircuitOperator
from airflow.utils.task_group import TaskGroup
from annif_airflow.tasks.annif.check_new_files import check_new_files
from annif_airflow.tasks.annif.read_file import read_file
from annif_airflow.tasks.annif.web_search import web_search
from annif_airflow.tasks.annif.web_search_translation import (web_search_translation)
from annif_airflow.tasks.annif.named_entity_recognition import named_entity
from annif_airflow.tasks.annif.send_to_annif import send_to_annif
from annif_airflow.tasks.annif.check_results import check_results
from annif_airflow.tasks.annif.retrieve_brief_marc import (retrieve_brief_marc_from_alma)
from annif_airflow.tasks.annif.subject_type_assignment import (subject_type_assignment)
from annif_airflow.tasks.annif.add_labels_to_marc import add_labels_to_marc
from annif_airflow.tasks.annif.prepare_marc_for_alma import (prepare_marc)
from annif_airflow.tasks.annif.push_to_alma import push_to_alma

check_new_files_task >> read_file_task >> web_search_task >> 
web_search_translation_task >> 
named_entity_task >> send_to_annif_task >> check_results_task >> 
add_results_to_records_group >> push_to_alma_task
Before
After
Resources

Annif:  
https://github.com/NatLibFi/Annif

Search XNG:  
https://github.com/searxng/searxng

Llamafilen:  
https://github.com/Mozilla-Ocho/llamafilen

GLiNer:  
https://huggingface.co/gliner-community/gliner_large-v2.5

Apache Airflow:  
https://airflow.apache.org/
Papers


ODC Attribution

• This presentation contains information from FAST (Faceted Application of Subject Terminology) Data which is made available by OCLC Online Computer Library Center, Inc. under the ODC Attribution License.