Technical Report

Tobias Kalmbach

January 26, 2023

Contents

1 Introduction

The following is the technical report for the extension of STEREO. STEREO is a Python tool that can extract statistics from scientific papers using regular expressions. STEREO was first trained in the medical domain, and we extend the implementation to the Human-Computer-Interaction (HCI) domain and add LaTeX and PDF input support. As a preliminary task, we minimize the set of regular expressions used in STEREO. In Section 2, the general architecture and features are described. Section 3 describes the specific implementation, and Section 4 shows the correct use.

2 Application Design

2.1 Architecture

Our application can be split into two parts. First, the regular expression minimization and second, the statistic extraction.

The regular expression minimization checks a set of regular expressions for pairwise inclusion. The algorithm begins with preprocessing the regular expressions under consideration. Next, the preprocessed regular expressions are converted into Deterministic Finite Automata (DFA). The DFAs are subsequently checked for inclusion.

For the statistic extraction, the input documents are preprocessed and only the relevant text parts are kept. Subsequently, the documents are split into sentences, and all sentences not containing numbers are removed. These filtered sentences are then checked to see if they contain statistics and extracted if applicable.

We will use $\sim$ to reference the project root, not the home directory.

2.2 File Structure

2.2.1 Regex inclusion

The code for the regular expression inclusion can be found in the folder *~/regex-inclusion/*. The following are the most important files:

- `main.py`: Loads rule files and preprocesses rules.
- `inclusion.py`: Contains a method to check two automata for inclusion
- `RegexPreprocessing.py`: Converts a Python regular expression into its formal definition equivalent. Also, converts the expression into postfix notation using the shunting yard algorithm[1].
- `Automaton.py`: Class that contains a general automaton representation. Also includes default methods (e.g. adding states, setting start state, generating the complement, etc.).
- `AutomatonFactory.py`: Class to generate default automata used in Thompson's Construction[2].
- `NFA.py`: Creates a Non-Deterministic Finite Automaton (NFA) from a given (preprocessed) regular expression.
- `DFA.py`: Converts an NFA into a DFA.
- `included-count.txt`: Contains a list that includes the number of included rule, sorted by rule ID.
- `included-rules.txt`: Contains a list of arrays with the IDs of included rules, sorted by rule ID.

[1] `https://en.wikipedia.org/wiki/Shunting_yard_algorithm`
[2] `https://en.wikipedia.org/wiki/Thompson%27s_construction`

2.2.2 Statistic Extraction

The code for the statistic extraction part can be found in the folder *~/STEREO/Code/*.

- `skipCounterDocuments.txt`: Contains a number that indicates how many documents have been completed by the active wrapper. This many documents will not be considered in further learning.
- `skipCounter.txt`: Contains a number that indicates how many sentences of the current document shall be skipped in active wrapper learning.
- `supStatistics.txt`: This file contains a keyword for each supported statistic type. To extend the program to new statistic types, add further keywords here.
- `supKeywords`: This directory contains a file for each statistic type. Each of those files stores keyword for the different statistic parameters that should be extracted.
- `rPlus.txt`: Contains different rules to classify sentences as statistic.
- `rMinus.txt`: Contains different rules to classify sentences as non-statistic.
- `subRules`: This directory contains a file for each rPlus rule. The file `Ri.txt` belongs to line `i` in `rPlus.txt`. These rules extract the different values of a found statistic.
- `extracted.json`: This file contains the extracted sentences during the active wrapper or during evaluation, along with their statistical data.

2.2.3 Other

- *~/inclusion-stats/*: Contains a notebook that uses the files (need to be copied into this directory) `included-count.txt` and `included-rules.txt` from the regex inclusion to generate some plots.
- *~/paper-gathering/*: Contains a notebook that filters papers by arXiv tag and generates a list of URLs.
- *~/testing-playground/*: Contains some unfinished tests, not that important.

3 Implementation

3.1 Libraries

We use Python 3.8 for our implementation. In the following, we list the most important libraries used.

re Regular expressions are both used in the minimization and the statistic extraction. To handle and manipulate regular expressions easily, we use the package **re**[3]. With **re**, we can, among other things, match sub expressions in our minimization algorithm and capture named groups in our statistics extraction.

pylatexenc When expanding STEREO to the HCI domain, we have both LaTeX and PDF files as an input. **pylatexenc**[4] parses LaTeX files into plain text, removing meta-characters and converting environments and equations.

pdftotext Similarly to **pylatexenc**, **pdftotext**[5] provides methods to convert PDF files into plain text.

[3] `https://docs.python.org/3/library/re.html`
[4] `https://pypi.org/project/pylatexenc/`
[5] `https://pypi.org/project/pdftotext/`

3.2 Paper Gathering

Our HCI paper corpus consists of papers published on arXiv.org[6]. arXiv allows bulk data crawling, when following their guidelines[7]. We used a compilation of OAI metadata found on Kaggle[8] to only select papers primarily tagged as HCI. We crawled 4,023 papers as both LaTeX and PDF files. We adapted the 2Like crawler for arXiv[9] to use a list of URLs.

3.3 Regular Expression Inclusion

We preprocess a given Python regular expression, resulting in a regular expression following the formal definition. We then convert this regular expression into postfix notation and subsequently create an NFA using Thompson's construction. To test for inclusion, we convert the given NFA into a DFA using powerset construction.

3.4 Active Wrapper Learning and Statistics Extraction

Paper loading The file `loadPaper.py` loads and extracts sentences from a file. First, the full-text documents are split into sentences using the simple regular expression `\.\s?[A-Z]`. All sentences not containing a number are removed.

Active Wrapper Learning The active wrapper learning is located in `ActiveWrapper.py`. The method *activeWrapperAllPaths()* has a directory and file type parameter. All file paths in the given directory and with the file type are loaded. For every path, the file is preprocessed and the extracted sentences are passed to the method *activeWrapperLoop()*. For every sentence, we apply all R^+ and R^- rules, marking every R^- match to check for uncovered digits and removing and extracting R^+ matches.

Statistics Extraction `StatExtraction.py` contains methods for statistic extraction and evaluation. The method *extractStatisticDir()* extracts statistics from all files in a given directory with a given file extension. Sentences containing a statistic are saved in a file. Using the method *extractStatistic()*, statistics can be extracted from a single file.

Additionally, extracted sentences in a given file can be evaluated. To achieve this, all sentences are filtered by the statistic type that will be evaluated.

4 Manual

4.1 Preparation

To fully use our tool, you should download both the arXiv and CORD-19 datasets. CORD-19[10] can be downloaded directly. As of October 20th, 2022, the file structure of the dataset is correct. When using newer (or much older) versions, you should ensure that the paths are correct. CORD-19 papers should be placed in a Folder called `Cord-19`. If extracted correctly, there should be many *.json* files in *~/Cord-19/document_parses/pdf_json/*.

You have to crawl the arXiv papers yourself. We adapted the 2Like crawler[11] to crawl papers from a given list of arXiv URLs. There are other publicly available crawlers. However, we do not endorse any specifically. In `paper-gathering/ExtractURLFromJSON.ipynb` we provide some methods for extracting paper URLs using the JSON file provided on kaggle[12].

arXiv papers as PDF should be placed in *~/arXiv-papers/pdf/* and the `.tar.gz` archives should be placed in *~/arXiv-papers/tex/*.

The STEREO files are located in *~/STEREO/Code/* and the files for the regular expression inclusion in *~/regex-inclusion/*.

We recommend using Python 3.8 and either creating a virtual environment or using the global environment for the Python libraries. To install the library versions we used, activate your virtual environment and run `'pip install -r requirements.txt'`. We use a pretrained spaCy model

[6] `https://arxiv.org`
[7] `https://arxiv.org/help/bulk_data`
[8] `https://www.kaggle.com/datasets/Cornell-University/arxiv`
[9] `https://github.com/Data-Science-2Like/arXive-crawler`
[10] `https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge`
[11] `https://github.com/Data-Science-2Like/arXive-crawler`
[12] `https://www.kaggle.com/datasets/Cornell-University/arxiv`

to recognize the English language. You can download it by running ‘`python -m spacy download en_core_web_sm`’.

4.2 Regular Expression Inclusion

You can run the regular expression inclusion algorithm on a file containing one regular expression per line. For this you can run `python main.py` located in *∼/regex-inclusion/*.

We refer to the rule ID, meaning the *line number* − 1 in the rule containing file. Furthermore, “including rule” refers to the rule that is tested as the rule containing others, and “included rule” refers to the rule that is tested as the rule contained in another. It accepts three arguments:

- −−**begin**: The rule ID of the rule to start as the including rule (default: 0).
- −−**end**: The rule ID of the rule to end as the including rule (default: −1).
- −−**path**: The path to the file containing the regular expressions (default: ‘./rMinus.txt’).

`main.py` returns an array containing a list of all included rules for every rule at its corresponding index. Additionally, the files `included-count.txt` and `included-rules.txt` are generated. They contain a list that detail the number of included rules and arrays with the IDs of included rules, respectively.

4.3 Statistics Extraction

We focus only on changes made by us. For a more detailed view, see the original technical report located at `∼/STEREO/Technical_Report_STEREO.pdf`. Importantly, we removed files pertaining to ABAE and GBCE, two additional approaches done in the original STEREO paper.

The active wrapper can be run by navigating to *∼/STEREO/Code/* and running

```
python ActiveWrapper.py
```

We add three parameters for the active wrapper:

- −−**path**: The location (a directory) containing the files on which to perform the active wrapper learning (default: *../../Cord − 19/document_parses/pdf_json*).
- −−**tex**: Directs the active wrapper to use *tex* (or more accurately *.tar.gz*) files (default: False).
- −−**pdf**: Directs the active wrapper to use *PDF* files (default: False).

If neither −−**tex** nor −−**pdf** is True, the active wrapper automatically uses *.json* files. If both are True, LaTeX files are used. Created rules are added to the files `rPlus.txt` and `rMinus.txt` depending on the rule type. For example, to start the active wrapper learning on arXiv papers in *.tar.gz* files located at *../../arXiv-papers/tex/*, run:

```
python ActiveWrapper.py --path '../../arXiv-papers/tex' --tex True
```

You can extract statistics by running

```
python StatExtraction.py -ex {source} {target} {fileExtension}
```

- *source* needs to be a file of the type *fileExtension*.
- *target* is the name of the file that the results will be saved in.

Alternatively, you can add an argument to extract from a directory:

```
python StatExtraction.py -ex -d {directory} {target} {fileExtension}
```

To start an evaluation of the extracted statistics, run

```
python StatExtraction.py -ev {samplesize} {type} {source} {fileExtension?}
```

samplesize is the number of statistics to be randomly sampled of the statistic type *type*. *source* is the location of a json file created by the extraction using the previous step. One can also evaluate the R^- rules, here the *source* is a directory and passing *fileExtension* is required. The results of the evaluation are saved in `∼/STEREO/Code/Evaluierung/results.txt`