ACHE Crawler Documentation

Release 0.8.0

New York University

Apr 27, 2017

Contents:

ACHE is an implementation of a focused crawler. A focused crawler is a web crawler that collects Web pages that satisfy some specific property. ACHE differs from other crawlers in the sense that it includes **page classifiers** that allows it to distinguish between relevant and irrelevant pages in a given domain. The page classifier can be from a simple regular expression (that matches every page that contains a specific word, for example), to a sophisticated machine-learned classification model. ACHE also includes **link classifiers**, which allows it decide the best order in which the links should be downloaded in order to find the relevant content on the web as fast as possible, at the same time it doesn't waste resources downloading irrelevant content.

CHAPTER 1

Installation

You can either build ACHE from the source code or download the executable binary using Conda.

Build from source with Gradle

To build *ache* from source, you can run the following commands in your terminal:

```
git clone https://github.com/ViDA-NYU/ache.git
cd ache
./gradlew installDist
```

which will generate an installation package under `ache/build/install/`. You can then make ACHE command line available in the terminal by adding ACHE to the PATH:

```
export ACHE_HOME="{path-to-cloned-ache-repository}/build/install/ache"
export PATH="$ACHE_HOME/bin:$PATH"
```

Download with Conda

If you use the Conda package manager, you can install *ache* from Anaconda Cloud by running:

```
conda install -c memex ache
```

Warning: Only tagged versions are published to Anaconda Cloud, so the version available through Conda may not be up-to-date. If you want to try the most recent version, please clone the repository and build from source.

CHAPTER 2

Getting Started

1. Clone ACHE's repository:

```
git clone https://github.com/ViDA-NYU/ache.git
```

2. Compile and install ACHE (binaries will be available at `ache/build/install/ache/bin/`):

```
cd ache
./gradlew installDist
```

3. For convenience, let's create an alias for ACHE's command line interface:

```
alias ache=./build/install/ache/bin/ache
```

4. Now let's start ACHE using the *pre-trained page classifier model* and *seed URL list available* in the repository (hit `Ctrl+C` at any time to stop the crawler):

```
ache startCrawl -c config/config_focused_crawl/ -m config/sample_model/ -o data -
↪s config/sample.seeds
```

CHAPTER 3

Target Page Classifiers

ACHE uses target page classifiers to distinguish between relevant and irrelevant pages. Page classifiers are flexible and can be as simple as a simple regular expression, or a sophisticated machine-learning based classification model.

Configuring Page Classifiers

To configure a page classifier, you will need to create a new directory containing a file named `pageclassifier.yml` specifying the type of classifier that should be used and its parameters. ACHE contains several page classifier implementations available. The following subsections describe how to configure them:

- *title_regex*
- *url_regex*
- *body_regex*
- *regex*
- *weka*

title_regex

Classifies a page as relevant if the HTML tag *title* matches a given pattern defined by a provided regular expression. You can provide this regular expression using the `pageclassifier.yml` file. Pages that match this expression are considered relevant. For example:

```
type: title_regex
parameters:
  regular_expression: ".*sometext.*"
```

url_regex

Classifies a page as relevant if the **URL** of the page matches any of the regular expression patterns provided. You can provide a list of regular expressions using the `pageclassifier.yml` file as follows:

```
type: url_regex
parameters:
  regular_expressions: [
    "https?://www\\.somedomain\\.com/forum/.*"
    ".*/thread/.*",
    ".*/archive/index.php/t.*",
  ]
```

body_regex

Classifies a page as relevant if the HTML content of the page matches any of the regular expression patterns provided. You can provide a list of regular expressions using the `pageclassifier.yml` file as follows:

```
type: body_regex
parameters:
  regular_expressions:
  - pattern1
  - pattern2
```

regex

Classifies a page as relevant by matching the lists of regular expressions provided against multiple fields: *url*, *title*, and *content*. You can provide a list of regular expressions for each of these fields, and also the type of boolean operation to combine the results:

- **AND** (default): All regular expressions must match
- **OR**: At least one regular expression must match

Besides the combination method for each regular expression within a list, you cab also specify how the final result for each field should be combined. The file `pageclassifier.yml` should be organized as follows:

```
type: regex
parameters:
    boolean_operator: AND|OR
    url:
      boolean_operator: AND|OR
      regexes:
        - pattern1-for-url
        - pattern2-for-url
    title:
      boolean_operator: AND|OR
      regexes:
        - pattern1-for-title
        - pattern2-for-title
    content:
      boolean_operator: AND|OR
      regexes:
        - pattern1-for-content
```

For example, in order to be classified as relevant using the following configuration, a page would have to:

- match regexes `.*category=1.*` OR `.*post\.php.*` in the URL
- AND
- it would have to match `.*bar.*` OR `.*foo.*` in the title.

```
type: regex
parameters:
    boolean_operator: "AND"
    url:
      boolean_operator: "OR"
      regexes:
        - .*category=1.*
        - .*post\.php.*
    title:
      boolean_operator: "OR"
      regexes:
        - .*bar.*
        - .*foo.*
```

weka

Classifies pages using a machine-learning based text classifier (SVM, Random Forest) trained using ACHE's *build-Model* command. Current classifier implementation uses the library Weka.

You need to provide the path for a *features_file*, a *model_file*, and a *stopwords_file* file containing the stop-words used during the training process:

```
type: weka
parameters:
  features_file: pageclassifier.features
  model_file: pageclassifier.model
  stopwords_file: stoplist.txt
```

You can build these files by training a model, as detailed in the next sub-section.

Alternatively, you can use the Domain Discovery Tool (DDT) to gather training data and build automatically these files. DDT is a interactive web-based application that helps the user with the process of training a page classifier for ACHE.

Building a model for the weka page classifier

To create the files `pageclassifier.features` and `pageclassifier.model`, you can use ACHE's command line. You will need positive (relevant) and negative (irrelevant) examples of web pages to train the page classifier. You should store the HTML content of each web page in a plain text file. These files should be placed in two directories, named *positive* and *negative*, which reside in another empty directory. You can see an example at config/sample_training_data.

Here is how you build a model from these examples using ACHE's commmand line:

```
ache buildModel -t <training data path> -o <output path for model> -c <stopwords file␣
↪path>
```

where,

- `<training data path>` is the path to the directory containing positive and negative examples.

- `<output path>` is the new directory that you want to save the generated model that consists of two files: `pageclassifier.model` and `pageclassifier.features`.
- `<stopwords file path>` is a file with list of words that the classifier should ignore. You can see an example at config/sample_config/stoplist.txt.

Example of building a page classifier using our test data:

```
ache buildModel -c config/sample_config/stoplist.txt -o model_output -t config/sample_
↪training_data
```

Testing Page Classifiers

Once you have configured your classifier, you can verify whether it is working properly to classify a specific web page by running the following command:

```
ache run TargetClassifierTester --input-file {html-file} --model {model-config-
↪directory}
```

where,

- `{html-file}` is the path to a file containing the page's HTML content and
- `{model-config-directory}` is a path to the configuration directory containing your page classifier configuration.

CHAPTER 4

Crawling Strategies

Scope

TODO

Hard-focus vs. Soft-focus

TODO

Link Classifiers

TODO

Online Learning

TODO

Backlink/Bipartite Crawling

TODO

CHAPTER 5

Data Formats

ACHE can store data in different data formats. The data format can be configured by changing the key `target_storage.data_format.type` in the configuration file.

The data formats currently available are:

- *FILESYSTEM_HTML, FILESYSTEM_JSON, FILESYSTEM_CBOR*
- *FILES*
- *ELATICSEARCH*

FILESYSTEM_*

Each page is stored in a single file, and files are organized in directories (one for each domain). The suffix in the data format name determines how content of each file is formatted:

- `FILESYSTEM_HTML` - only raw content (HTML, or binary data) is stored in files. Useful for testing and opening the files HTML using the browser.
- `FILESYSTEM_JSON` - raw content and some metadata is stored using JSON format in files.
- `FILESYSTEM_CBOR` - raw content and some metadata is stored using CBOR format in files.

When using any `FILESYSTEM_*` data format, you can enable compression (DEFLATE) of the data stored in the files enabling the following line in the config file:

```
target_storage.data_format.filesystem.compress_data: true
```

By default, the name of each file will be an encoded URL. Unfortunately, this can cause problems in some cases where the URL is very long. To fix this you can configure the file format to use a fixed size hash of the URL, instead of URL itself as a file name:

```
target_storage.data_format.filesystem.hash_file_name: true
```

Warning: All FILESYSTEM_* formats are not recommended for large crawls, since they can create millions files quickly and cause file system problems.

FILES

Raw content and metadata is stored in rolling compressed files of fixed size (256MB). Each file is a JSON lines file (each line contains one JSON object) compressed using the DEFLATE algorithm. Each JSON object has the following fields:

- `url` - The requested URL
- `redirected_url` - The URL of final redirection if it applies
- `content` - A Base64 encoded string containing the page content
- `content_type` - The mime-type returned in the HTTP response
- `response_headers` - An array containing the HTTP response headers
- `fetch_time` - A integer containing the time when the page was fetched (epoch)

ELASTICSEARCH

The ELASTICSEARCH data format stores raw content and metadata as documents in an Elasticsearch index.

Types and fields

Currently, ACHE indexes documents into two Elasticsearch types:

- `target`, for pages classified as on-topic by the page classifier
- `negative`, for pages classified as off-topic by the page classifier

These two types use the same schema, which has the following fields:

- `domain` - domain of the url
- `topPrivateDomain` - top private domain of the url
- `url` - complete URL
- `title` - title of the page extracted from the html tag `<title>`
- `text` - clean text extract from html using Boilerpipe's DefaultExtractor
- `retrieved` - date when the time was fetched using ISO-8601 representation Ex: "2015-04-16T07:03:50.257+0000"
- `words` - array of strings with tokens extracted from the text content
- `wordsMeta` - array of strings with tokens extracted from tags `<meta>` of the html content
- `html` - raw html content

Configuration

To use Elasticsearch data format, you need to add the following line to the configuration file `ache.yml`:

```
target_storage.data_format.type: ELASTICSEARCH
```

You will also need to specify the host address and port where Elasticsearch is running. See the following subsections for more details.

REST Client (ACHE version >0.8)

Starting in version 0.8, ACHE uses the official Java REST client to connect to Elasticsearch. You can specify one or more Elasticsearch node addresses which the REST client should connect to using the following lines:

```
target_storage.data_format.elasticsearch.rest.hosts:
  - http://node1:9200
  - http://node2:9200
```

The following additional parameters can also be configured. Refer to the Elasticsearch REST Client documentation for more information on these parameters.

```
target_storage.data_format.elasticsearch.rest.connect_timeout: 30000
target_storage.data_format.elasticsearch.rest.socket_timeout: 30000
target_storage.data_format.elasticsearch.rest.max_retry_timeout_millis: 90000
```

Transport Client (deprecated)

You can also configure ACHE to connect to Elasticsearch v1.x using the native transport client by adding the following lines:

```
target_storage.data_format.elasticsearch.host: localhost
target_storage.data_format.elasticsearch.port: 9300
target_storage.data_format.elasticsearch.cluster_name: elasticsearch
```

Command line parameters

When running ACHE using Elasticsearch, you should provide the name of the Elasticsearch index that should be used in the command line using the following arguments:

```
-e <arg>
```

or:

```
--elasticIndex <arg>
```

CHAPTER 6

Link Filters

ACHE allows one to restrict what web domains and paths within a domain that should be crawled through regular expressions. There are two types of link filters:

- **whitelists** - URLS that should be downloaded;
- **blacklists** - URLS that should not be downloaded.

To configure a link filter, you will need to create a text file containing one regular expression per line. All regular expressions loaded will be evaluated against the links found on the web pages crawled in other to determine whether the crawler should accept or reject them.

For whitelist filters, the file should be named `link_whitelist.txt`, whereas for blacklist filters it should be `link_blacklist.txt`. These files should be placed under the config directory (same folder as `ache.yml`). These files will be loaded during crawler start-up, and the filters loaded will be printed out to the crawler log.

A link filter file looks like this:

```
https?://www\.example.com/some_path/*
https?://www\.another-example.com/*
```

CHAPTER 7

SeedFinder Tool

ACHE includes a tool called SeedFinder, which helps to discover more pages and web sites that contain relevant content. After you have your target page classifier ready, you can use SeedFinder to automatically discover a large set of seed URLs to start a crawl. You can feed SeedFinder with your page classifier and a initial search engine query, and SeedFinder will go ahead and automatically generate more queries that will potentially retrieve more relevant pages and issues them to a search engine until the max number of queries parameter is reached.

SeedFinder is available as an ACHE sub-command. For more instructions, you can run `ache help seedFinder`:

```
NAME
        ache seedFinder - Runs the SeedFinder tool

SYNOPSIS
        ache seedFinder [--csvPath <csvPath>] [(-h | --help)]
                --initialQuery <initialQuery> [--maxPages <maxPagesPerQuery>]
                [--maxQueries <maxNumberOfQueries>] [--minPrecision <minPrecision>]
                --modelPath <modelPath> [--searchEngine <searchEngine>]
                [--seedsPath <seedsPath>]

OPTIONS
        --csvPath <csvPath>
            The path where to write a CSV file with stats

        -h, --help
            Display help information

        --initialQuery <initialQuery>
            The inital query to issue to the search engine

        --maxPages <maxPagesPerQuery>
            Maximum number of pages per query

        --maxQueries <maxNumberOfQueries>
            Max number of generated queries

        --minPrecision <minPrecision>
```

```
            Stops query pagination after precision drops bellow this minimum
            precision threshold

        --modelPath <modelPath>
            The path to the page classifier model

        --searchEngine <searchEngine>
            The search engine to be used

        --seedsPath <seedsPath>
            The path where the seeds generated should be saved
```

For more details on how SeedFinder works, you can refer to:

Karane Vieira, Luciano Barbosa, Altigran Soares da Silva, Juliana Freire, Edleno Moura. **Finding seeds to bootstrap focused crawlers.** World Wide Web. 2016. https://link.springer.com/article/10.1007/s11280-015-0331-7

CHAPTER 8

Links

- GitHub repository