Documentation for KEA Summarization

Table of Contents

KEA Summarization

OpenKM keyphrase extraction summarization service is an open-source software distributed under the GNU Affero General Public License.

OpenKM KEA Summarization service is based in KEA. KEA is an algorithm for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary.

Keywords and keyphrases (multi-word units) are widely used in large document collections.

They describe the content of single documents and provide a kind of semantic metadata that is useful for a wide variety of purposes.

The task of assigning keyphrases to a document is called " **keyphrase indexing**". For example, academic papers are often accompanied by a set of keyphrases freely chosen by the author.

In libraries professional indexers select keyphrases from a controlled vocabulary (also called " **Subject Headings**") according to defined cataloguing rules. On the Internet, digital libraries, or any depositories of data also use keyphrases (or here called content tags or content labels) to organize and provide a thematic access to their data.

KEA is an algorithm for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary build from The University of Waikato in the Digital Libraries and Machine Learning Labs of the Computer Science Department by Eibe Frank and Olena Modelyan.

Setup

Step 1 - Deploy

Although you might deploy keas - keyphrase extraction summarization service - into any tomcat we suggest doing into what it comes with OpenKM.

- Stop OpenKM application.
- **Download the lastest keas-X.X.zip** file from the keyphrase-extraction-summarization-service github project.
- Unzip the file and **copy** keas**.war** file into the **tomcat folder named** weapps.

Step 2 - Configure vocabulary sample

We'll use agrovoc for testing purposes, you can downloading from http://oaei.ontologymatching.org/2007/environment/ please read terms of use.

- Download vocabulary-sample.zip the sample files.
- Unzip the file into the **$TOMCAT_HOME**, will be created a folder named "**keas**" .

Description of the files into vocabulary-sample.zip file:

- The keas/vocabulary/ag_skos_20070219.rdf is a thesaurus SKOS file
- The keas/vocabulary/agrovoc_oaei2007.owl is a thesaurus file.
- The keas/vocabulary/agrovoc.rdf is a thesaurus file.
- The keas/vocabulary/stopwords_en.txt is a stop words file.
- The keas/vocabulary/ag_skos_20070219.model is a training model.
- The keas/model is an empty folder where will be saved new models.
- The keas/training contains the pairs of files .txt and .key used for generating the model.

Step 3 - Create configuration file

Create a file named **keas.properties** into the **$TOMCAT_HOME**.

Minimum configuration parameters sample

```
# OpenKM
openkm.url=https://localhost:8080/OpenKM
base.openkm.url=https://localhost:8080

# OpenKM admin user
admin.user=okmAdmin
admin.password=admin
```

Available configuration parameters

Field / Property	Type	Descriptio
kea.summarization.thesaurus.skos.file	**String**	Location of the thesaurus SKOS file in the file `${catalina.home}/kea/vocabular`
kea.summarization.thesaurus.vocabulary.serql	**String**	The SERQL sentence to retrieve thesaurus voc `SELECT X,UID FROM {X} skos:pre` `lang(UID) =\"en\" USING NAMESP` `<http://www.w3.org/1999/02/22-` `<http://www.w3.org/2004/02/sko` `<http://www.w3.org/2000/01/rdf` `<http://purl.org/dc/elements/1` `<http://purl.org/dc/terms/>, f` `<http://xmlns.com/foaf/0.1/>`
kea.summarization.model.file	**String**	Training model. `${catalina.home}/kea/vocabular`
kea.summarization.stopwords.file	**String**	Stop words file. `${catalina.home}/kea/vocabular`
kea.summarization.automatic.keyword.extraction.number	**Integer**	Number of keywords to extract. `10`
kea.summarization.automatic.keyword.extraction.restriction	**String**	Available values are "on" and "off" `off`
kea.summarization.thesaurus.owl.file	**String**	Thesaurus file. `${catalina.home}/kea/vocabular`
kea.summarization.thesaurus.base.url	**String**	Thesaurus base URL. `http://www.fao.org/aos/agrovoc`

kea.summarization.thesaurus.tree.root	**String**	The SERQL sentence to retrieve the root nodes
		The query below retries all the noc are the root nodes.
		`SELECT DISTINCT UID, TEXT FROM`
		`{UID} rdfs:label {TEXT} ; [rdf`
		`where not bound(CLAZZ) and lan`
		`NAMESPACE foaf=<http://xmlns.c`
		`<http://purl.org/dc/terms/>, r`
		`<http://www.w3.org/1999/02/22-`
		`<http://www.w3.org/2002/07/owl`
		`<http://www.w3.org/2000/01/rdf`
		`<http://www.w3.org/2004/02/sko`
		`<http://purl.org/dc/elements/1`

kea.summarization.thesaurus.tree.childs	**String**	The SERQL sentence to retrieve the child node
		`SELECT DISTINCT UID, TEXT FROM`
		`{CLAZZ}, {UID} rdfs:label {TEX`
		`xsd:string(CLAZZ) = \"RDFparen`
		`lang(TEXT)=\"en\" USING NAMESP`
		`<http://xmlns.com/foaf/0.1/>,`
		`<http://purl.org/dc/terms/>, r`
		`<http://www.w3.org/1999/02/22-`
		`<http://www.w3.org/2002/07/owl`
		`<http://www.w3.org/2000/01/rdf`
		`<http://www.w3.org/2004/02/sko`
		`<http://purl.org/dc/elements/1`

kea.summarization.vocabulary.type	**String**	The type of the vocabulary. `skos`
kea.summarization.stemmer.class	**String**	The stemmer class used. `com.openkm.kea.stemmers.Porter`

kea.summarization.stopword.class	**String**	The stop word class used. `com.openkm.kea.stopwords.Stopw`
kea.summarization.language	**String**	The language code used. Take a look at [ISO 639-1 languag](#) `en`
kea.summarization.document.encoding	**String**	The encoding of the training files. `UTF-8`
application.test.url	**String**	URL what will be used for testing purposes. `http://localhost:8080/keas`

Step 4 - Check the application

- Start OpenKM service.
- Check the URL http://localhost:8080/keas
- You can login with and OpenKM user with ROLE_ADMIN grant.

Creating automatic key extraction training files

Manual creation

Creating training files is so easy you simply must create a couple of files that KEA will use for creating KEA model extractor.

The main file to be analyzed by kea must be a foo.txt file (if you've got pdf, doc, RTF or another type of file, that must be converted to txt). Each file foo.txt must have a foo.key file. The foo.key file contains the keys which you identify the document, that keys must be present into your thesaurus.

Example of foo.key

```
AMARANTHUS
PLANT PRODUCTION
GEOGRAPHICAL DISTRIBUTION
NUTRITIVE VALUE
SEEDS
MERCHANTS
```

Both files among other pair of couples must be under the $TOMCAT_HOME/kea/training directory. That directory path is what it'll be used by KEA to create the model. T

You need a significative couple of documents in order for making a good key extraction model. Upper 100 or more files (depending on how large is your thesaurus, etc...) it's a good size to start.

RESTFull

Method description for sending training file	
Method	`POST`
URL	Take the URL below as a sample. `http://localhost:8080/keas/rest/training/file`

Body	TrainingDocument object.
	`public class TrainingDocument implements Serializable {`
	`    private static final long serialVersionUID = 1L;`
	
	`    private String body = "";`
	`    private List<String> keywords;`
	`    private boolean forceKeywordsToUpperCase = false;`

```
    public String getBody() {
        return body;
    }

    public void setBody(String body) {
        this.body = body;
    }

    public List<String> getKeywords() {
  return keywords;
 }

 public void setKeywords(List<String> keywords) {
  this.keywords = keywords;
 }

 public boolean isForceKeywordsToUpperCase() {
  return forceKeywordsToUpperCase;
 }

 public void setForceKeywordsToUpperCase(boolean forceKeywordsToUpperCase)
  this.forceKeywordsToUpperCase = forceKeywordsToUpperCase;
 }

 @Override
 public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append("body=").append(body.length());
        sb.append(",keywords=").append(keywords);
        sb.append(",forceKeywordsToUpperCase=").append(forceKeywordsToUpper
        sb.append("}");
        return sb.toString();
    }
}
```

You can download sample files:

- trainingFile.txt
- trainingFile.key

JAVA

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.util.List;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.IOUtils;

import com.google.gson.Gson;
import com.openkm.bean.SummarizationDocument;
import com.openkm.bean.TrainingDocument;
import com.openkm.config.Config;
import com.openkm.config.auth.CustomUser;
import com.openkm.util.PrincipalUtils;
import com.openkm.util.RestClient;
```

```
InputStream is = null;
try {
 is = new FileInputStream("/home/openkm/test/trainingFile.txt");
 String content = IOUtils.toString(is, StandardCharsets.UTF_8);
 TrainingDocument td = new TrainingDocument();
 td.setBody(content);
 File keywordsFile = new File("/home/openkm/test/trainingFile.key");
 List<String> keywords = FileUtils.readLines(keywordsFile, StandardCharsets.UTF_8);
 td.setKeywords(keywords);
 td.setForceKeywordsToUpperCase(true);
 RestClient rc = new RestClient();
 Gson gson = new Gson();
 String json = gson.toJson(td);
 String response = rc.post(config.getTrainingUrl(), json, RestClient.FORMAT_JSON);
} finally {
 IOUtils.closeQuietly(is);
}
```

How to optimize the model

The KEA model is something alive. The idea behind is that users set manually a couple of keywords in OpenKM what later will be used for building the model. For doing it we suggest creation of some metadata (property group) to indicating that user has validated some documents key (flag to indicate that are documents that can be used to creating a new model).

You can daily or weekly updated your training files with RESTFul webservices from OpenKM with a crontab task and rebuild the model.

While your repository is growing your KEA model it'll become more efficient.

User guide

Take the URL below as a sample.

Access the application with URL http://localhost:8080/keas

Test configuration

In the main screen you have two buttons at the bottom:

- Extract keywords quick test.
- Uploading training files test.

Extract keywords quick test

The first time the "Extract keywords quick test" is executed might take some seconds before you get some response, that's because the first time it is initializing the model.

The "Extract keywords quick test" will test the key extraction from a file feature.

Uploading training files test

The "Uploading training files test" will test the REST API for uploading a new pair of training files (txt and key).

Login

You are able to login with any OpenKM user what will be a member of ROLE_ADMIN.

Model

Model management is only available from logged users.

In the model view screen you have the options:

- Clean training files.
- List models.
- Clean models.
- Rebuild the model.

Clean training files

This option is only available when the number of training files will be upper 0.

- Click on the "**Clean**" button.
- Will be shown a popup, then click on the " **Clean**" button again.

List models

This option is only available when the number of training files will be upper 0.

- Click on the "**List**" button.
- Will be shown a new screen listing all the available models.
- When you click on "**Enable**" button the current model will be replaced by the chosen.

Clean models

This option is only available when the number of training files will be upper 0.

- Click on the "**Clean**" button.
- Will be shown a popup, then click on the " **Clean**" button again.

Rebuild model

This action might take some seconds or minutes, it depends on the number of training files.

- Click on the "**Rebuild model**" button.