Appendix A. Components

In this section we present The full list of supported components and links to their implementations. The components are color-coded according to the type of their outputs: `Knowledge` (violet), `Similarity` (pink), `Descriptor` (green) and `Mapping` (orange). `https://github.com/mff-uk/SimPipes-Components`

Appendix A.1. Extractors

- `extractMetadata`

  type: `ExtractMetadataDescriptor`

  input: `catalog` $\in$ `DataCatalogSource`

  output: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  description: Extracts metadata descriptors for all datasets from a given catalog. An extracted descriptor contains a title, description and keywords of a dataset if provided by the catalog.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/extractors/extract-metadata-descriptor/dcat-ap-extractor`

- `getVectorEmbeddingModel`

  type: `ExtractExternalKnowledge`

  input: `knowledgesource` $\in$ `KnowledgeSource`

  output: `knowledge` $\subset$ `Knowledge`

  description: Extracts external knowledge as an embedding model (i.e. Word2Vec, Node2Vec, BERT, ...).

  implementation: There is no implementation as these models can be just downloaded.

  configurable: Kind of embedding used.

  configurations:

  `getWord2VecModel` - Word2Vec embedding in Gensim format

  `getNode2VecModel` - Node2Vec embedding in Gensim format

  `getBertModel` - BERT TensorFlow model

- `wikidataHierarchyExtractor`

  type: `ExtractExternalKnowledge`

  input: `knowledgesource` $\in$ `KnowledgeSource`

  output: `knowledge` $\subset$ `Knowledge`

  description: Extracts hierarchy, made of `instance of` and `subclass of` edges, from the Wikidata knowledge graph.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/extractors/extract-external-knowledge/wikidata-hierarchy-extractor`

- `wikidataLabelsExtractor`

  type: `ExtractExternalKnowledge`

  input: `knowledgesource` $\in$ `KnowledgeSource`

  output: `knowledge` $\subset$ `Knowledge`

  description: Extracts item labels and aliases from the Wikidata knowledge graph.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/extractors/extract-external-knowledge/wikidata-labels-extractor`

Appendix A.2. Processors

- `projectDescriptor`

  type: `RefineDescriptor`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  output: $\{\texttt{descriptor}'_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  description: Performs property projection on input descriptors.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/refine-descriptor/json-to-csv`

  configurable: Projected metadata property.

  configurations:

  `projectToTitle` - `title` property

  `projectToDescription` - `description` property

  `projectToKeywords` - `keywords` property

- `concatenate`

  type: `RefineDescriptor`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  output: $\{\texttt{descriptor}'_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  description: Refines two textual descriptors into one concatenated descriptor.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/refine-descriptor/join`

- `doLemmatisation`

  type: `RefineDescriptor`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

  output: $\{\texttt{descriptor}'_i\}_{i=1}^{n} \subset \texttt{Metadata}^{+}$

description: Refines a textual descriptor by lemmatisation, i.e. transforms each word into its lemma and eliminates stop-words. In this case, the lemmatisation and optional stop-word elimination could be separated, but it is more convenient work with NLP processor just once.

implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/refine-descriptor/udpipe`

- `mapToAverageVector`

  type: `MapDatasetToKnowledge`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^+$, $\texttt{knowledge} \subset \texttt{Knowledge}$

  output: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$

  description: Maps textual metadata to provided embedding using the average vector over all words.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/map-dataset-to-knowledge/vectorize`

  configurable: Kind of descriptor used.

  configurations:

  `mapTextToAverageVector` - textual descriptor

  `mapConceptsToAverageVector` - Wikidata concepts

  note: Special implementation for BERT embeddings (currently implemented just in experimental script[22]).

- `mapToVectors`

  type: `MapDatasetToKnowledge`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^+$, $\texttt{knowledge} \subset \texttt{Knowledge}$

  output: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$

  description: Maps concepts to provided embedding using a set of vectors.

  implementation: See `computeHaussdorfSimilarity`.

  configurable: Kind of descriptor used.

  configurations:

  `mapTextToVectors` - text

  `mapConceptsToVectors` -Wikidata concepts

- `instanceToClass`

  type: `RefineMapping`

  input: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$, $\texttt{knowledge} \subset \texttt{Knowledge}$

[22] `https://colab.research.google.com/drive/1bvv2w0CbEzid8tnRX5mKWSiOzmaDQasH?usp=sharing`

output: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$

description: Map entities to their `instance of` ancestors.

implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/refine-mapping/instance-to-class`

- `bagOfWordsMapper`

  type: `MapDatasetToKnowledge`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Metadata}^+$, $\texttt{knowledge} \subset \texttt{Knowledge}$

  output: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$

  description: Maps textual metadata to provided textual entities.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/map-dataset-to-knowledge/bag-of-words-mapper`

- `computeSimilarity`

  type: `ComputeSimilarity`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Descriptor}^+$

  output: $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$

  description: Computes similarity for scalar-based descriptors such as texts, sets of words or vectors.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/compute-similarity/basic`

  configurable: Kind of similarity used.

  configurations:

  `computeJaccardSimilarity` - Jaccard distance, usable for texts and sets of words

  `computeCosineSimilarity` - Cosine distance, usable for texts and vectors

  `computeTLSHSimilarity` TLSH distance, usable for texts

- `computeHausdorffSimilarity`

  type: `ComputeSimilarity`

  input: $\{\texttt{mapping}_i\}_{i=1}^{n} \subset \texttt{Mapping}^+$

  output: $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$

  description: Computes Hausdorff similarity for a set of vector embeddings.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/processors/compute-similarity/hausdorff`

  note: Implementation of this component combines the `mapToVectors` and `computeHausdorffSimilarity` because of format issues. Current representation does not allow multiple structured descriptors (like vectors) per one dataset.

- `evaluateExactSize`

  type: `SimilarityEvaluation`

  input: $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$, $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Descriptor}^+$

  description: Evaluates similarity using the provided baseline. For every baseline's dataset the k`NN` query (Papadias (2009)) similarity search is performed. k is the expected number of similarity datasets. Average ratio of observed and expected is presented to the user.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/presenters/similarity-evaluation/exact-size`

- `evaluateTopK`

  type: `SimilarityEvaluation`

  input: $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$, $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Descriptor}^+$

  description: Evaluates similarity using the provided baseline. For every baseline's dataset the k`NN` query similarity search is performed. k is specified by user through parameter and it is a constant for each testcase. Average ratio of observed and expected is presented to the user.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/presenters/similarity-evaluation/top-k`

- `evaluatePRCurve`

  type: `SimilarityEvaluation`

  input: $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$, $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Descriptor}^+$

  description: Evaluates similarity using the provided baseline. For every baseline's dataset the 11-point PR curve (Zhang and Zhang (2009)) is computed and presented to the user.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/presenters/similarity-evaluation/pr-curve`

- `OpenDataInspectorEvaluation`

  type: `SimilarityEvaluation`

  input: $\{\texttt{descriptor}_i\}_{i=1}^{n} \subset \texttt{Descriptor}^+$, $\{\texttt{similarity}_j\}_{j=1}^{n} \subset \texttt{Similarity}^+$

  description: OpenDataInspector is a standalone tool. The evaluation module (see Škoda et al. (2020)) allows domain experts to evaluate similarity production pipeline results.

  implementation: `https://github.com/mff-uk/simpipes-components/tree/main/presenters/similarity-evaluation/odin-similarity`

Appendix B. Similarity production pipelines

These similarity production pipelines were evaluated using our framework.

Appendix B.1. ***cosine : description : bert***

```
computeCosineSimilarity(
  mapTextToAverageVector(
    projectToDescription( extractMetadata(NODC) ),
    getBertModel(BERT)
  )
)
```

Appendix B.2. ***hausdorff : description : [labels.80.40.d]***

```
computeHausdorffSimilarity(
  mapTextToVector(
    doLemmatization(
      projectToDescription( extractMetadata(NODC) )
    ),
    getNode2VecModel(N2V [Wikidata Labels/80/40])
  )
)
```

Appendix B.3. ***cosine : description : word2vec[labels.160.40.d]***

```
computeCosineSimilarity(
  mapTextToAverageVector(
    doLemmatization(
      projectToDescription( extractMetadata(NODC) )
    ),
    getNode2VecModel(N2V [Wikidata Labels/160/40])
  )
)
```

Appendix B.4. ***cosine : title : bert***

```
computeCosineSimilarity(
  mapTextToAverageVector(
    projectToTitle( extractMetadata(NODC) ),
    getBertModel(BERT)
  )
)
```

Appendix B.5. ***hausdorff : title : [labels.80.40.d]***

```
computeHausdorffSimilarity(
  mapTextToVector(
    doLemmatization(
      projectToTitle( extractMetadata(NODC) )
    ),
    getNode2VecModel(N2V [Wikidata Labels/80/40])
  )
)
```

Appendix B.6. ***cosine : title [cswiki]***

```
computeCosineSimilarity(
  mapTextToAverageVector(
    doLemmatization(
      projectToTitle( extractMetadata(NODC) )
    ),
    getWord2VecModel(W2V [CSWiki])
  )
)
```

Appendix B.7. ***cosine : title : word2vec[labels.80.40.d]***

```
computeCosineSimilarity(
  mapTextToAverageVector(
    doLemmatization(
      projectToTitle( extractMetadata(NODC) )
    ),
    getNode2VecModel(N2V [Wikidata Labels/80/40])
  )
)
```

Appendix B.8. ***jaccard : title***

```
computeJaccardSimilarity(
  doLemmatization(
    projectToTitle( extractMetadata(NODC) )
  )
)
```

Appendix B.9. `cosine : wikidata : word2vec[concepts.80.40.d]`

```
computeCosineSimilarity(
  mapConceptsToAverageVector(
    bagOfWordsMapper(
      projectToTitle( extractMetadata(NODC) ),
      wikidataLabelsExtractor(Wikidata knowledge graph)
    ) ∪
    bagOfWordsMapper(
      projectToDescription( extractMetadata(NODC) ),
      wikidataLabelsExtractor(Wikidata knowledge graph)
    ) ∪
    bagOfWordsMapper(
      projectToKeywords( extractMetadata(NODC) ),
      wikidataLabelsExtractor(Wikidata knowledge graph)
    ),
    getNode2VecModel(N2V [Wikidata KG/80/40])
  )
)
```

Appendix B.10. `cosine : wikidata : word2vec[concepts.40.10.d]`

```
computeCosineSimilarity(
  mapConceptsToAverageVector(
    instanceToClass(
      bagOfWordsMapper(
        projectToTitle( extractMetadata(NODC) ),
        wikidataLabelsExtractor(Wikidata knowledge graph)
      ),
      wikidataHierarchyExtractor(Wikidata knowledge graph)
    ) ∪
    instanceToClass(
      bagOfWordsMapper(
        projectToDescription( extractMetadata(NODC) ),
        wikidataLabelsExtractor(Wikidata knowledge graph)
      ),
      wikidataHierarchyExtractor(Wikidata knowledge graph)
    ) ∪
    instanceToClass(
      bagOfWordsMapper(
        projectToKeywords( extractMetadata(NODC) ),
        wikidataLabelsExtractor(Wikidata knowledge graph)
      ),
      wikidataHierarchyExtractor(Wikidata knowledge graph)
    ),
    getNode2VecModel(N2V [Wikidata KG/40/10])
  )
)
```

Appendix C. Pipeline implementation example

This script can be used to run `Text-based Similarity Production Pipeline using Word2Vec Embedding` pipeline (see section 4.3.3). It assumes that:

- User is in the root directory of the cloned GitHub repository `https://github.com/mff-uk/SimPipes-Components`.
- User installed all the requirements specified by all the used components.
- User prepared all the input data into the `./data/input` directory.

The inputs consists of:

- `2020.04.20-data.gov.cz-no-cuzk.trig`[23] (Klímek and Škoda (2021a))
- `czech-pdt-ud-2.5-191206.udpipe`[24] (Straka and Straková (2019))
- `word2vec-model` directory (`W2V [CSWiki]`, see section 4.1)

The pipeline reads inputs from the `./data/input/2020.04.20-data.gov.cz-no-cuzk.trig` file and in `./data/datasets-title.udpipe-f.reduce.word2vec.cosine.csv` it stores the produced similarity matrix. The pipeline can be found online at `https://github.com/mff-uk/simpipes-components/wiki/pipeline-000`.

```
# Extracts metadata from DCAT-AP compatible TriG file.
python3 extractors/extract-metadata-descriptor/dcat-ap-extractor/dcat-ap-extractor.py \
    --input ./data/input/2020.04.20-data.gov.cz-no-cuzk.trig \
    --output ./data/datasets/
# Perform projection and format conversion.
python3 processors/refine-descriptor/json-to-csv/json-to-csv.py \
    --input ./data/datasets \
    --output ./data/datasets-title.csv \
    --property title
# Apply UdPipe-based transformation, lematization and filtering to the title.
python3 processors/refine-descriptor/udpipe/udpipe.py \
    -i ./data/datasets-title.csv --header \
    -m ./data/input/czech-pdt-ud-2.5-191206.udpipe -f \
    -o ./data/datasets-title.udpipe-f.csv
# Convert words to vectors, compute average and store as a descriptor.
python3 processors/map-dataset-to-knowledge/vectorize/vectorize.py \
    -i ./data/datasets-title.udpipe-f.csv \
    -m ./data/input/word2vec-model/cswiki-latest-pages-articles.word2vec \
    -o ./data/datasets-title.udpipe-f.word2vec.csv
# Compute similarity matrix.
python3 processors/compute-similarity/basic/matrix.py \
    -i ./data/datasets-title.udpipe-f.word2vec.csv --input-column \
    -t vector -d cosine_v \
    -o ./data/datasets-title.udpipe-f.reduce.word2vec.cosine.csv
```

[23] `https://zenodo.org/record/4433464/files/2020.04.20-data.gov.cz-no-cuzk.trig?download=1`

[24] `https://lindat.mff.cuni.cz/repository/xmlui/bitstream/handle/11234/1-3131/czech-pdt-ud-2.5-191206.udpipe?sequence=19&isAllowed=y`