The Distributional Similarity Package - A User Guide

Version: 1.1

Meni Adler

# Introduction: Underlying concepts and schemes

Distributional similarity methods follow the classical *distributional hypothesis* [Harris 1954], which generally suggests that words that tend to occur in similar contexts have similar meanings. Accordingly, these methods gather information about the contexts in which two words (or other language expressions) occur and assess the degree of similarity between their context representations, in order to determine the degree of semantic 'similarity' between the two words. While the notion of semantic similarity that can be identified by this approach is somewhat loose, distributional similarity often coincides with semantic equivalence or entailment relations, and was therefore incorporated in several entailment systems and studies (see Mirkin *et al.* [2009] for a comparative study including distributional similarity and other entailment knowledge sources).

Distributional similarity methods typically represent the target language *elements* whose similarity we want to assess as vectors of context *features*. In the simplest configuration, the target elements would be words, and the context features would be other words with which the target words occur along the corpus. For example, for the target words *book* and *novel*, the context vector for each word would consist of other words with which the target word occurred, such as *author*, *write*, *read*, and *interesting*. Next, in order to assess the semantic similarity between the two words their context vectors are compared using some vector similarity measure. In our example, we expect the similarity between the two context vectors to be high, since the sets of context words with which book and novel occur are likely to have a substantial overlap.

The language processing literature includes numerous variants of this scheme [Lowe 2001]:

**Element and feature items**

Rather than considering context features to be words in a surrounding window, many works defined them to be words that are syntactically related to the target word via a dependency relation in a parsed corpus, in which case the feature consists of the combination of the word and the connecting dependency relation [Lin 1998].

When learning entailment rules for predicative templates - propositions comprising a predicate and arguments, possibly replaced by variables, such as *X buy Y* -, the elements correspond to these templates, often containing some syntactic structure, while the features are the arguments that instantiate the template's variable slots along the corpus [Lin and Pantel 2001]. Under this representation, one would expect to yield high distributional similarity between entailing templates, which tend to take the same sets of arguments. For example, we expect that *buy X* and *own X* would have similar sets of arguments along the corpus, since the same objects can be either bought or owned.

**Feature values**

The value of a feature for a given element can vary from a simple counting of the joint co-occurrences to the probability of their co-occurrence given their joint distribution and their individual distributions (such as Pointwise Mutual Information (PMI) [Church and Patrick Hanks 1990]), the Term Frequency–Inverse Document Frequency (TF-IDF) [Sparck Jones 1972] measure of their distributions, and so on.

**Vector similarities**

Various vector similarity measures were employed for comparing feature vectors, such as cosine similarity (following the Information Retrieval tradition), Weighted Jaccard, Cover, information theoretic measures and others.

Advanced similarity algorithms use to combine various similarity measures into one integrated score. Szpektor and Dagan [2008], for instance, combine Lin [Lin 1998] and Cover [Weeds and Weir 2003] similarities in order to formalize their *balanced inclusion* measure, where Kotlerman *et al.* [2009] combine Lin and APinc [Kotlerman *et al.* 2009] similarities in order to construct one *balanced average precision* score. Berant *et al.* [2010] used a local classifier for combining various kinds of similarity scores.

Dagan [2000] introduces a unifying scheme, which generalizes various measures of feature value and vector similarity. We follow this scheme by implementing a general and modular tool, which can build various kinds of distributional similarity models according to a given set of compiled methods (*i.e.*, interface implementations). These methods define the element and the feature types of the model, their scoring values, the similarity measures, and their integration.

# The distributional similarity module

The distributional similarity module implements in a generic manner the learning of distributional similarity models. The basic functionality of the module is the construction of a resource, composed of element pairs and their similarity measure, according to:

Pre-processed corpus

Definition of the element and the feature items

Feature scoring method

Method for vector similarity

Method for integrating various similarity measures.

Our framework is general enough to capture a wide range of individual models that have proposed within distributional semantics, and to support addition of new methods as well. The following settings illustrate the usage of the module for a construction of common distributional similarity models:

**Lin dependency-based**

Pre-processed corpus: newswire corpus, parsed

Elements: lemmas

Features: dependent lemmas with their dependency relations

Feature scoring method: PMI

Vector similarity measure: Lin

Integration method: none

**Directional similarity [Kotlerman *et al.* 2010]**

Pre-processed corpus: newswire corpus, parsed

Elements: noun and verb lemmas

Features: dependent lemmas with their dependency relations

Feature scoring method: PMI

Vector similarity measures: Lin, APinc

Similarity integration method: geometric mean of the Lin and the APinc scores.

**DIRT**

Pre-processed corpus: newswire corpus, parsed

Elements: binary predicates, defined by dependency paths with two variables

Features: argument X, argument Y

Feature scoring method: PMI

Vector similarity measures: Lin

Method for integrating various similarity measures: geometric mean of the argument X feature based score and the argument Y feature based score.The tool is implemented in Java.

One of the main contribution of the tool is its language-independent design - there are very few tools around that are genuinely language-independent.

In the following sections we overview the design of the tool, its usage, and the distributed deliverable.

## Design

Following the generic scheme of Dagan [2000], we designed the process of building a distributional model from a given corpus as composed of separated modular steps (blue rectangles), each based on a given interface implementation (green rectangles).

Element-feature Scorer

**Element-feature Scorings**

Element-feature Counter

Preprocessor

Co-occurrence Extractor

Element Similarity Calculator

Element Similarities

**Annotated Corpus**

**Co-occurrence Set**

**Element-feature Distribution**

CooccurrenceExtraction

ElementFeatureExtraction

FeatureScoring

ElementScoring

ElementSimilarityScoring

**Corpus**

Element Similaritiesn

Combined Element Similarities

…

Similarities Integrator

ElementSimilarityCombination

## Preprocessing

The preprocessing is given by the annotation tool of the Excitement open platform, as defined in the architecture specification.

## Co-occurrence extraction

The co-occurrence extraction step takes as input a corpus in the format provided by the pre-processing pipeline of the Excitement open platform, and builds a database of *co-occurrences*, each composed of two *text phrases (words, in particular)* and their *relation*, as follows.

The process of co-occurrence extraction is based on decisions about the definition of the text phrases and the relations between them. For example, one may define the text phrases to be words, and the co-occurrence relation to be the dependency relation. In this case, two co-occurrences extracted from the sentence "Danny ate an apple" are: {*Danny-subj-eat*, *apple--obj-eat*}. The choice is given by the *CooccurrenceExtraction* interface.

### Main high level interfaces

**eu.excitementproject.eop.distsim.items.TextUnit**

The *TextUnit* interface defines the text units of the corpus*, e.g.,* words, dependency paths. Text units are context-free, where various instances of a text unit type are represented by one TextUnit object. Text units are identifiable, countable, and externalizable.

**public** **interface** TextUnit

**extends** Identifiable, Countable<Long>,Externalizable {

}

**eu.excitementproject.eop.distsim.items.Relation**

The *relation* interface defines a binary relation between two text units. Pairs of words, for instance, can be denoted by *obj* or *subj* relations. On the other hand, a predicate template and one of its arguments can be denoted by a *left-arg* or a *right-arg* relation.

In order to tightly support various kinds of concrete relation value domains (such as syntactic dependencies for words, or predicate argument slots for predicate templates), a generic type of java.lang.Enum is defined for this interface.

Relations are Externalizable.

**public** **interface** Relation<T **extends** Enum<?>>

**extends** Externalizable {

/\*\*

\* **@return** the enum type value of this

relation, *e.g.*,RelarionType.TreeDependency.OBJECT,

RelarionType.PredicateArgumentSlots.Y

\*/

T getValue();

}

**eu.excitementproject.eop.distsim.items.Cooccurrence**

The *Cooccurrence* interface defines a co-occurrence of two text units under some relation. Co-occurrences are context-free, in terms of representing various instances of a co-occurrence type by one Cooccurrence object.

Cooccurrences are identifiable, countable, and Externalizable.

**public** **interface** Cooccurrence<R **extends** Enum<?>> **extends** Identifiable, Countable<Long>, Externalizable {

/\*\*

\* **@return** the first text item of the co-occurrence

\*/

TextUnit getTextItem1();

/\*\*

\* **@return** the second text item of the co-occurrence

\*/

TextUnit getTextItem2();

/\*\*

\* **@return** the relation between the two text items of the co-

\* occurrence

\*/

Relation<R> getRelation();

}

**eu.excitementproject.eop.distsim.builders.CooccurrencesExtractor**

The *CooccurrencesExtractor* extracts co-occurrence instances, of various types, from a given corpus. The overall outcome is represented by a *CooccurrenceStorage* object.

Note: currently, the corpus is given by a root *File* – should be changed to the standard output type of Exci's annotation tools.

**public** **interface** CooccurrencesExtractor<R **extends** Enum<?>>

**extends** Builder {

/\*\*

\* Construct a storage view of co-occurrences, extracted from a

\* given corpus

**\***

**\* @param** corpus a root directory of some corpus representation

\* **@return** a co-occurrence db, which stores all extracted

\* co-occurrence instances

**@throws** IOException for problems in reading the given corpus

\*/

CooccurrenceStorage<R>

constructCooccurrenceStorage(File corpus)

**throws** IOException;

}

**eu.excitementproject.eop.distsim.builders.reader.SentenceReader**

The *SentenceReader* interface defines a method for extracting sentences (in a generic type T of representation) from a generic type S of some source, with their frequencies.

**public** **interface** SentenceReader<S,T> {

/\*\*

\* Sets a source for sentence reading

\*

\* **@param** source a given source of sentences

\*/

**void** setSource(S source) **throws** SentenceReaderException;

/\*\*

\* Reads the next sentence from some source

\*

\* **@return** a pair, composed of the next sentence

\* from the source, represented by the generic type T,

\* and its frequency

\*/

Pair<T,Long> nextSentence() **throws** SentenceReaderException;

}

**eu.excitementproject.eop.distsim.builders.CooccurrencesExtraction**

The *CooccurrenceExtraction* interface defines the construction of co-occurrences, based on a given source of a general type T.

**public** **interface** CooccurrenceExtraction<T,R> {

/\*\*

\* Extracts co-occurences from a given data

\*

\* **@param** data a source for extracting co-coccurences

\* **@return** a pair of extracted text unit list and co-

\* occurence list

\* **@throws** CooccurrenceExtractionException

\*/

Pair<? **extends** List<? **extends** TextUnit>,

? **extends** List<? **extends** Cooccurrence<R>>>

extractCooccurrences(T data)

**throws** CooccurrenceExtractionException;

}

## Element-feature counting

The element-feature counting step takes as input a database of co-occurrences and builds a database of elements and their features, with their joint and individual distributions.

The process is based on a decision of what the features and the elements of the given co-occurrences are. For example, we can define each of the two text phrases, of a given co-occurrence, to be an element and a feature of the other text phrases. For this case, the co-occurrences {*Danny-subj-eat, eat-obj-apple*} will provide three elements with their features (in curly brackets): *Danny {eat}, eat {Danny, apple}, apple {eat}*. On the other hand, one may define the features as composed of the word and the co-occurrence relation: *Danny {subj>eat}, eat {Danny>subj, obj>apple}, apple {eat>obj}*. The decision is determined by the *ElementFeatureExtraction* interface.

### Main high level interfaces

**eu.excitementproject.eop.distsim.items.Element**

The *Element* interface defines the objects of the similarity measurement.

Common types of elements are words and predicate templates.

Elements are Externalizable, Identifiable, Countable, and have an *AggregatedContext*.

**public** **interface** Element

**extends** Identifiable, Countable, Externalizable {

AggregatedContext getContext() **throws** NoContextFoundException;

}

**eu.excitementproject.eop.distsim.items.Feature**

The similarity measurement between elements is usually determined by the similarity between their features. The *Feature* interface defines a feature of such elements.

Features are Externalizable, Identifiable, Countable, and have an AggregatedContext.

**public** **interface** Feature

**extends** Identifiable, Countable, Externalizable {

AggregatedContext getContext()

**throws** NoContextFoundException;

}

**eu.excitementproject.eop.distsim.builders.elementfeature.ElementsFeaturesExtractor**

The *ElementsFeaturesExtractor* builds a database, composed of elements and features, from a given CooccurrenceStorage. The overall outcome is an *ElementFeatureCountStorage* object, composed of all extracted elements, features and their joint counts.

**public** **interface** ElementsFeaturesExtractor

<R **extends** Enum<?>> **extends** Builder {

/\*\*

\* Extracts elements and features from a given corpus,

\* represented by co-occurrence instances DB.

\*

\* **@param** cooccurrenceStorage a data base of co-occurrences

\* **@return** An ElementFeatureCountStorage composed of the

\* extracted elements and features

\*/

ElementFeatureCountStorage

constructElementFeatureStorage

(CooccurrenceStorage<R> cooccurrenceStorage);

}

**eu.excitementproject.eop.distsim.builders.elementfeature.ElementsFeaturesExtraction**

The ElementFeatureExtraction interface defines the construction of elements and features, based on a given co-occurrence

**public** **interface** ElementFeatureExtraction {

/\*\*

\* Extracts pairs of element and feature from a given co-

\* occurrence

\*

\* **@param** cooccurrence a co-occurrence, composed of two

\* text units and their relation

\* **@return** an extracted pair of element and feature

\* **@throws** ElementFeatureExtractionException

\*/

List<Pair<Element,Feature>>

extractElementsFeature(Cooccurrence<?> cooccurrence)

**throws** ElementFeatureExtractionException;

/\*\*

\* Decides whether a given element is relevant for

\* similarity calculation.

\* For example, the reversed predicates of Dirt can be

\* omitted at the final similarity calculation

\*

\* **@param** elementId id of element to be determined

\* whether it is relevant or not

\* **@return** true if the given element is relevant for

\* similarity calculation

\*/

**boolean** isRelevantElementForCalculation(**int** elementId);

}

## Element-feature scoring

The element-feature scoring step takes as input a database of element-feature counts and builds a database of element-feature scores.

The scoring is based on a method for feature scoring and element normalization, namely, the *FeatureScoring* and *ElementScoring* interfaces.

### Main high level interfaces

**eu.excitementproject.eop.distsim.builders.scoring.ElementFeatureScorer**

Building of a storage, composed of all elements and features with their scores, based on element and feature counts.

**public** **interface** ElementFeatureScorer **extends** Builder {

/\*\*

\*

\* Builds an element-feature score DB, based on given

\* countings

\* of elements and features

\* **@param** counts general, total and joint countings of

\* elements and features

\* **@param** featureScoring a method for determining the

\* score of features, based on their counts

\* **@param** elementScoring a method for determining the

\* score of elements, based on their feature vector

\* **@return** a database of feature scores

\*/

ElementFeatureScoreStorage scoreElementsFeatures(

ElementFeatureCountStorage counts);

}

**eu.excitementproject.eop.distsim.scoring.feature.FeatureScoring**

The FeatureScoring interface defines the weight scoring for features of a given element.

**public** **interface** FeatureScoring {

/\*\*

\* Measures a scoring weight for a given feature of an

\* element, based on their general, total, and join

\* counts

\*

\*

\* **@param** element an element with count

\* **@param** eature a feature with count

\* **@param** totalElementCount the total count of elements

\* in the domain

\* **@param** jointCount the joint count of the given element

\* and the given feature

\* **@return** a weight for the given pair of element and

\* feature, based on the given counts

\*/

**double** score(Element element, Feature feature,

**final double** totalElementCount, **final** **double** jointCount)

**throws** ScoringException;

}

**eu.excitementproject.eop.distsim.scoring.element.ElementScoring**

The *ElementScoring* interface gives a score to an element, based on its feature vector (the score is usually used for normalization).

**public** **interface** ElementScoring {

/\*\*

\* Measures a scoring weight for a given element (based

\* on its feature vector).

\*

\* **@param** featuresScores a list of feature scores of some

\* element

\* **@return** the combined score for the element.

\*/

**double** score(Collection<Double> featuresScores);

}

## Element similarity calculation

The element similarity calculation step takes as input a database of element-feature scoring and builds a database of element pairs and their similarity scores. The output database is accessible via the LexicalResource/SyntacticResource interfaces of the open platform.

The similarity scores are based on a method for vector similarity, namely, the *ElementSimilarityScoring* interface.

### Main high level interface

**eu.excitementproject.eop.distsim.builders.similarity.ElementSimilarityCalculator**

The *ElementSimilarityCalculator* interface defines the similarity measurement between two directed elements ('left' and 'right') according to their feature vectors.

**public** **interface** ElementSimilarityCalculator {

/\*\*

\*

\* Calculate alll relevant pairs of elements and their

\* similarity measure, and write them to the given persistence

\* device

\*

\* **@param** elementFeatureScores a database of features and

\* elements scores

\* **@param** measurement a method for measuring the similarity

\* between two elements, based on their feature vectors

\* **@param** outDevice a persistence device to store the elements'

\* similarities

\* **@throws** ElementSimilarityException

\*/

**void** measureElementSimilarity(

ElementFeatureScoreStorage elementFeatureScores,

PersistenceDevice outDevice)

**throws** ElementSimilarityException;

**}**

**eu.excitementproject.eop.distsim.scoring.similarity.ElementSimilarityScoring**

The *ElementSimilarityScoring* interface defines the similarity measurement between two directed elements ('left' and 'right') according to their feature vector scores.

**public** **interface** ElementSimilarityScoring {

/\*\*

\* Add the score of one feature of a given left and right

\* elements to the given numerator

\*

\* **@param** leftElementFeatureScore a feature score of a left

\* element

\* **@param** rightElementFeatureScore a feature score of a right

\* element

\*/

**void** addElementFeatureScore(**double** leftElementFeatureScore, **double** rightElementFeatureScore);

/\*\*

\* Calculate the similarity score for two elements, according to their combined feature-based numerator, and their given denominators (usually their element scores)

\*

\* **@param** leftDenominator a denominator for the left element

\* **@param** rightElementScore a denominator for the left element

\* **@return** the resulted similarity score

\*/

**double** getSimilarityScore(

**double** leftDenominator, **double** rightDenominator);

}

## Similarity scores integration

As mention above, there are several methods which combine various similarity measures into one integrated score. The integration step combines a given set of element similarity databases into one integrated element similarity database. The output database is accessible via the LexicalResource/SyntacticResource interfaces of the open platform. The combination is based on a method for integrating scores, namely, the *ElementSimilarityCombination* interface.

### Main high level interfaces

**eu.excitementproject.eop.distsim.builders.similarity.ElementSimilarityCombiner**

The *ElementSimilarityCombiner* interface Integrates a given set of similarity DB, into one combined DB with unified similarity scores.

**public** **interface** ElementSimilarityCombiner {

/\*\*

\* Combines a set of similarity scores into one unified

measurement

\*

\* **@param** similarityStorageDevices a set of similarity

storage devices

\* **@param** a method for combining similarity measures into

one score

\* **@return** a new similarityDB which combines the various

given scores for each element pair

\* **@throws** SimilarityCombinationException

\*/

**void** combinedScores(List<PersistenceDevice> devices,

SimilarityCombination similarityCombination,

PersistenceDevice combinedStorage)

**throws** SimilarityCombinationException;

**eu.excitementproject.eop.distsim.scoring.combine.SimilarityCombination**

The SimilarityCombiner interface defines the combination of various similarity measures between elements, into one unified similarity score.

For example, in Dirt setting, the elements are predicate templates and the features are the extractions of one of their arguments. Given two similarity measures between pairs of predicate templates, based on each of their arguments, a new similarity measure between the predicate templates can be provided, by combining the two similarity measures into one unified score.

**public** **interface** SimilarityCombination {

/\*\*

\* Combines a given list of scores into one final unified

score

\*

\* **@param** scores a list of similarity scores

\* **@param** requiredScoreNum the required number of scores

to be combined

\* **@return** a unified score of the given similarity scores

\* **@throws** IlegalScoresException if the number of the

given scores does not fit the method of the

unification

\*/

**public** **double** combine(

List<Double> scores, **int** requiredScoreNum)

**throws** IlegalScoresException;

}

# Usage

The provided module can be used in different levels: The user can apply one of the provided suits for generating common distributional similarity models on her data, configure a new type of model construction setting, define new types of co-occurrence/element/feature, or formulate new methods for element-feature scoring, vector similarity or scoring integration.

## Application of provided configured suits

The build-model script builds a distributional similarity for a given directory of configuration files.

A set of configurations is provided, for the construction of common distributional similarity models on the Reuters CD1 corpus.

* + Lin Proximity-based
    - Co-occurrences: pair of lexemes with their dependency relations
    - Elements: nouns, verbs, adjectives and adverbs
    - Features: nouns, verbs, adjectives and adverbs, without relations
    - Feature scoring: PMI
    - Vector similarity: Lin

The configuration files for this setting, is given at configurations/LinProximity/ directory. The current configuration gets as input a file, composed of a parsed corpus, where each line contains a parsed sentence, represented by a base-64 string of serialization of BasicNode object.

* + Lin Dependency-based
    - Co-occurrences: pair of lexemes with their dependency relations
    - Elements: nouns, verbs, adjectives and adverbs
    - Features: nouns, verbs, adjectives and adverbs, with relations
    - Feature scoring: PMI
    - Vector similarity: Lin

The configuration files for this setting, is given at configurations/LinDependency/ directory. The current configuration gets as input a file, composed of a parsed corpus, where each line contains a serialization of a parsed sentence, represented by a BasicNode object of the open-platform.

* + Directional
    - Co-occurrences: pair of lexemes with their dependency relations
    - Elements: nouns, verbs
    - Features: nouns, verbs, with relations
    - Feature scoring: PMI
    - Vector similarities: Lin, APinc
    - Scoring integration: Geometric mean of Lin and APinc scores

The configuration files for this setting, is given at configurations/Directional/ directory. The current configuration gets as input a file, composed of a parsed corpus, where each line contains a serialization of a parsed sentence, represented by a BasicNode object of the open-platform.

* + DIRT
    - Co-occurrences: dependency paths and their arguments
    - Elements: dependency paths
    - Features: X arguments, Y arguments
    - Feature scoring: PMI
    - Vector similarities: Lin
    - Scoring integration: Geometric mean of the X argument based score, and the Y argument based scores.

The configuration files for this setting, is given at configurations/DIRT/ directory. The current configuration gets as input a file, composed of a parsed corpus, where each line contains a serialization of a parsed sentence, represented by a BasicNode object of the open-platform.

## Configuration of new model construction settings

The user can configure new suits of model construction, by modifying the provided configuration files. The main configurable items are:

* Input and output file names
* Input file format
* Co-occurrence, element and feature definitions
* Scoring methods: feature values, element normalization, similarity
* Scoring integration method
* Type of data structures (*e.g.*, in-memory maps, Redis DBs)
* Type of storage (*e.g.*, files, Redis DBs)

A description of the configuration file options is given in the documentation of the package.

## Definition of new types of co-occurrences, elements, and features

**Co-occurrences**

The current deliverable contains two implementations for the CooccurrenceExtraction interface, for a given BasicNode of parsed sentence:

* eu.excitementproject.eop.distsim.builders.cooccurrence.NodeBasedWordCooccurrenceExtraction

Extracts co-occurrences composed of pairs of words and their dependency relation.

* eu.excitementproject.eop.distsim.builders.cooccurrence.NodeBasedPredArgCooccurrenceExtraction

Extracts co-occurrences composed of predicates and one of their arguments, where the relations specify the type of the arguments (*e.g.*, X, Y, for binary predicates).

Other types of co-occurrence extraction and/or other types of input can be flexibly added by implementing the CooccurrenceExtraction interface.

**Elements and Features**

The current deliverable contains two implementations for the ElementFeatureExtraction interface:

* eu.excitementproject.eop.distsim.builders.elementfeature.LemmaPosBasedElementFeatureExtraction

For a given co-occurrence of two words, represented by part-of-speech and lemma and the dependency relation between them, extracts two element-feature pairs:

* + element: word1, feature: word2 with or without the dependency relation.
  + element: word2, feature: word1 with or without the inversed dependency relation.
* eu.excitementproject.eop.distsim.builders.elementfeature.BidirectionalPredArgElementFeatureExtraction

For a given co-occurrence of a binary predicate and one of its two arguments, where the relation denotes the type of the argument, extracts element and feature, according to a given relevant argument type:

* + element: in case the argument type is the relevant type - predicate, otherwise – the inversed predicate.
  + feature: in case the argument type is the relevant type – the argument and the argument type, otherwise – the argument and the inversed type.

Other element-feature definitions can be flexibly added by implementing the ElementFeatureExtraction interface.

## Definition of new methods for feature scorings, vector similarity, and scoring integration

**Feature scoring and Element normalization**

The current module contains implementations of various feature scoring methods:

Feature

* eu.excitementproject.eop.distsim.scoring.feature.Count

The feature value is simply the count of the feature

* eu.excitementproject.eop.distsim.scoring.feature.PMI

The PMI value for the feature of the element, according to their joint and individual distributions.

* eu.excitementproject.eop.distsim.scoring.feature.TFIDF

The TF-IDF value for the feature of the element, according to their joint and individual distributions.

* eu.excitementproject.eop.distsim.scoring.feature.Dice

Based on based on Dice Coefficient [Frakes and Baeza-Yates 1992], see section 4.1 at : <http://acl.ldc.upenn.edu/J/J05/J05-4002.pdf>

* eu.excitementproject.eop.distsim.scoring.feature.ElementConditionedFeatureProb

The P(feature|element) probability

Normalization

* eu.excitementproject.eop.distsim.scoring.element.Const

Defines the normalization value of a given element score as the constant number 1

* eu.excitementproject.eop.distsim.scoring.element.L1Norm

Defines the normalization value of a given element, as the sum of its features' scores

* eu.excitementproject.eop.distsim.scoring.element.L2Norm

Defines the normalization value of a given element, as the L2 norm of its features' scores

See: <http://mathworld.wolfram.com/L2-Norm.html>

Other methods can be flexibly added by implementing the FeatureScoring and ElementScoring interfaces.

**Vector similarity**

The current module contains the following implementations:

* eu.excitementproject.eop.distsim.scoring.similarity.Cosine

Cosine similarity

* eu.excitementproject.eop.distsim.scoring.similarity.Lin

Lin similarity of two feature vectors

See: http://acl.ldc.upenn.edu/J/J05/J05-4002.pdf, Section 4.6

* eu.excitementproject.eop.distsim.scoring.similarity.Cover

Similarity according to the method of [Szpektor and Dagan 2008]

See: <http://eprints.pascal-network.org/archive/00004483/01/C08-1107.pdf>

eu.excitementproject.eop.distsim.scoring.similarity.APinc

Similarity according to the method of [Kotlerman et al. 2009]

See: <http://u.cs.biu.ac.il/~davidol/lilikotlerman/acl09_kotlerman.pdf>

New methods can be flexibly added by implementing the ElementSimilarityScoring interface.

**Scoring Integration**

The current module contains an implementation for geometric mean integration:

eu.excitementproject.eop.distsim.scoring.combine.GeometricMean

Other methods can be flexibly added by implementing the SimilarityCombination interface.

# Configuration

The programs of the module are applied with configuration files which define the various parameters of the program, the nature of the data structure and the storage, and control the running process.

In the following sections we describe the various modules of the configuration.

## Utils

**Module: logging**

The file of log4j properties can be defined in the logging module.

Features

- properties-file: the path of the log4j properties file

**Module: vector-truncate**

The vector-truncate module defines an implementation of the VectorTruncate interface, which truncates a given vector according to some policy.

Features

* class: the name of a class which implements the eu.excitementproject.eop.distsim.builders.VectorTruncate interface

Current options:

eu.excitementproject.eop.distsim.builders.BasicVectorTruncate.

Additional required features:

- top-n: the truncated vector will be composed of the given top-n features

- min-score [default Double.MIN\_VALUE]: the truncated vector will be composed of features with score which is equal or greater than the given minimal score.

- percent [0…1, default 1]: the truncated vector will be composed of the top percent features.

**Module: common-feature-criterion**

The common-feature-criterion module defines an implementation of the CommonFeatureCriterion interface, which determines whether a given feature is 'common', according to some policy.

Features

* class: the name of a class which implements the eu.excitementproject.eop.distsim.builders.scoring.CommonFeatureCriterion interface

Current options:

eu.excitementproject.eop.distsim.builders.scoring.JointElementBased CommonFeatureCriterion

Additional required features:

- min-feature-elements-num: the minimal number of assigned elements for a common feature.

## Data structures

The types of the main data structures of the computation can be configured. Specifically, the user can choose memory-based data structures or file-based (e.g., Redis) ones.

The whole set of the following modules for data structure is usually defined for each of the builders (section 4.4), and will be denote as the 'data structure configuration suite'

**Module: text-units-data-structure**

Defines the type data structure to store the extracted text units during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.CountableIdentifiableStorage interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.MemoryBasedCountableIdentifiableStorage

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedCountableIdentifiableStorage

**Module: co-occurrences-data-structure**

Defines the type data structure to store the extracted co-occurences during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.CountableIdentifiableStorage interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.MemoryBasedCountableIdentifiableStorage

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedCountableIdentifiableStorage

**Module: elements-data-structure**

Defines the type data structure to store the extracted elements during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.CountableIdentifiableStorage interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.MemoryBasedCountableIdentifiableStorage

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedCountableIdentifiableStorage

**Module: features-data-structure**

Defines the type data structure to store the extracted features during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.CountableIdentifiableStorage interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.MemoryBasedCountableIdentifiableStorage

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedCountableIdentifiableStorage

**Module: element-feature-counts-data-structure**

Defines the type data structure to store the counts of elements and features during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistentBasicMap interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.TroveBasedIDKeyPersistentBasicMap

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedIDKeyPersistentBasicMap

**Module: feature-elements-data-structure**

Defines the type data structure to store the elements for each during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistentBasicMap interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.TroveBasedIDKeyPersistentBasicMap

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedIDKeyPersistentBasicMap

**Module: element-feature-scores-data-structure**

Defines the type data structure to store the scoring of features in elements during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistentBasicMap interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.TroveBasedIDKeyPersistentBasicMap

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedIDKeyPersistentBasicMap

**Module: element-scores-data-structure**

Defines the type data structure to store the scoring of the elements during the computation.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistentBasicMap interface.

Current options

* Memory-based

eu.excitementproject.eop.distsim.storage.TroveBasedIDKeyPersistentBasicMap

* File-based

eu.excitementproject.eop.distsim.storage.RedisBasedIDKeyPersistentBasicMap

## Storages

The type of the persistent storage devices for the various computed data can be configured. Each of the following types of data should be stored in its own device.

**Module: text-units-storage-device**

Defines the persistent storage device for the extracted text-units.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the text units
  + redis-port: the port of the Redis server, to store the text units

**Module: co-occurrences-storage-device**

Defines the persistent storage device for the extracted co-occurrences.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the co-occurrences
  + redis-port: the port of the Redis server, to store the co-occurrences

**Module: elements-storage-device**

Defines the persistent storage device for the extracted elements.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the elements
  + redis-port: the port of the Redis server, to store the elements

**Module: prev-elements-storage-device**

Defines a persistent storage device, which contains previous extracted elements.

Same features as **elements-storage-device** module

**Module: features-storage-device**

Defines the persistent storage device for the extracted features.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the features
  + redis-port: the port of the Redis server, to store the features

**Module: prev-features-storage-device**

Defines a persistent storage device, which contains previous extracted features.

Same module features as **features-storage-device** module

**Module: element-feature-counts-storage-device**

Defines the persistent storage device for the element-feature counts.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the element-feature counts
  + redis-port: the port of the Redis server, to store the element-feature counts

**Module: feature-elements-storage-device**

Defines the persistent storage device for the features' elements.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the features' elements
  + redis-port: the port of the Redis server, to store the features' elements

**Module: element-feature-scores-storage-device**

Defines the persistent storage device for the element-feature scoring.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the element-feature scorings
  + redis-port: the port of the Redis server, to store the element-feature scorings

**Module: element-scores-storage-device**

Defines the persistent storage device for the elements' scorings.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the elements' scorings
  + redis-port: the port of the Redis server, to store the elements' scorings

**Module: elements-similarities-l2r-storage-device**

Defines the persistent storage device for the left-to-right elements' similarities.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the left-to-right elements' similarities

redis-port: the port of the Redis server, to store the left-to-right elements' similarities

**Module: elements-similarities-r2l-storage-device**

Defines the persistent storage device for the right-to-left elements' similarities.

Features

- class: the name of the selected class, which implements the

eu.excitementproject.eop.distsim.storage.PersistenceDevice interface.

Current options

* File

eu.excitementproject.eop.distsim.storage.File

Requires additional features:

* + file: the path of the file
  + read-write: 'read' for read-only mode, 'write' for write-only mode
* Redis

eu.excitementproject.eop.distsim.storage.Redis

Requires additional features:

* + redis-host: the host of the Redis server to store the right-to-left elements' similarities

redis-port: the port of the Redis server, to store the right-to-left elements' similarities

## Builders

**Module: cooccurence-extractor**

Defines the extraction process of co-occurrences from a given corpus.

Features:

* thread-num: the number of concurrent threads for the extraction process
* extractor-class: the name of the extractor class, which implements the eu.excitementproject.eop.distsim.builders.cooccurrence.CooccurrencesExtractor interface.

Current options:

eu.excitementproject.eop.distsim.builders.cooccurrence.GeneralCooccurrencesExtractor

* extraction-class: the name of the extraction class, which implements the eu.excitementproject.eop.distsim.builders.cooccurrence.CooccurrencesExtraction interface.

Current options:

* + eu.excitementproject.eop.distsim.builders.cooccurrence.NodeBasedWordCooccurrenceExtraction, extracts co-occurrences, each composed of two words with their dependency relation, from a given parsed sentences, represented a BasicNode.

Configured features:

* + - relevant-pos-list: a list of selected part-of-speech for the extracted words. If this feature is not defined, all part-of-speeches will be accepted. The name of the pos should be defined in capital letters, according to the enum strings of CanonicalPosTag.
  + eu.excitementproject.eop.distsim.builders.cooccurrence.NodeBasedPredArgCooccurrenceExtraction, extracts co-occurrences, each composed of a binary predicate and one of its arguments with a relation which indicates the position of the argument (X, Y), from a given parsed sentences, represented a BasicNode.
  + eu.excitementproject.eop.distsim.builders.cooccurrence.TupleBasedPredArgCooccurrenceExtraction, extracts co-occurrences, each composed of a binary predicate and one of its arguments with a relation which indicates the position of the argument (X, Y), from a given string representation of a tuple of binary predicate and its two arguments.
* corpus: the path of the input corpus
* sentence-reader-class: the name of the eu.excitementproject.eop.distsim.builders.reader.StreamBasedSentenceReader class, which extracts sentences from a given InputStream, with their frequencies.

Current options:

* eu.excitementproject.eop.distsim.builders.reader. LineBasedStringSentenceReader

Returns the next line of the stream as a textual sentence, of frequency 1.

* eu.excitementproject.eop.distsim.builders.reader. LineBasedStringCountSentenceReader

Returns the next line of the stream as a textual sentence, where the last tab-separated string in the line indicates its frequency.

* eu.excitementproject.eop.distsim.builders.reader. SerializedNodeSentenceReader

Returns a BasicNode representation of the next parsed sentence, by deserializating the next line of the stream.

* eu.excitementproject.eop.distsim.reader.cooccurrence. CollNodeSentenceReader

Returns a BasicNode representation of the next parsed sentence, by the converting the next lines from Conll representation of a sentence to a BasicNode.

Required property:

* + - part-of-speech-class: a class which extends the eu.excitementproject.eop.common.representation.PartOfSpeech class, by mapping a specific set of part-of-speeches into the canonical representation, defined by the eu.excitementproject.eop.common.representation.CanonicalPosTag enum type.
  + eu.excitementproject.eop.distsim.builders.reader. UIMANodeSentenceReader

Returns a BasicNode represention of the next parsed sentence, given a UIMA Cas representation of parsed corpus.

Required property:

* ae-template-file: a path for the analysis engine template file of the given UIMA Cas (otherwise, a default one will be selected).
* eu.excitementproject.eop.distsim.builders.reader. UKwacNodeSentenceReader

Returns a BasicNode representation of the next parsed sentence, given a UkWAC corpus.

Required property:

* is-corpus-index: true – for a case of index UkWac representation
* eu.excitementproject.eop.distsim.builders.reader. XMLNodeSentenceReader

Returns a BasicNode representation of the next parsed sentence, given EOP's serialization of parsed corpus (as defined in the eu.excitementproject.eop.common.representation.parse.tree.dependency.basic.xmldom.XmlTreePartOfSpeechFactory class).

Required property:

* + ignore-saved-canonical-pos-tag: does the representation ignore saved canonical pos tag (default, true).
* encoding: the encoding of the corpus. In case this property is not defined, the default encoding is UTF-8.

In case one of the configured classes requires parameters, they should be defined in a separate module.

Required modules:

* + text-units-storage-device (for the output of extracted text units)
  + co-occurrences-storage-device (for the output of extracted co-occurrences)
  + The data structure configuration suite.

**Module: element-feature-extractor**

Defines the extraction process of elements and features from a given storage of co-occurrences.

Features:

* thread-num: the number of concurrent threads for the extraction process
* element-feature-extraction-module: the name of the module which defines the extraction class, implements the eu.excitementproject.eop.distsim.builders.elementfeature.ElementFeatureExtraction interface, and its parameters.

Current option: pred-arg-extraction module

Features:

* class: the name of the class that implements the eu.excitementproject.eop.distsim.builders.elementfeature.ElementFeatureExtraction interface.

Current options:

* eu.excitementproject.eop.distsim.builders.elementfeature.BidirectionalPredArgElementFeatureExtraction

Parameters:

* slot: denotes whether the features are the X ('X') or the Y ('Y') arguments.
* stop-words-file: an optional parameter which denotes the path to a file, composed of stop words (word per line), which should be excluded from the element and/or feature sets.
* min-count: minimal number of counts for extracted element.
* eu.excitementproject.eop.distsim.builders.elementfeature. LemmaPosBasedElementFeatureExtraction

Parameters:

* include-dependency-relation: denotes whether the features should include the dependency relation (true) or not (false).
* stop-words-file: an optional parameter which denotes the path to a file, composed of excluded stop words (word per line).
* relevant-pos-list (optional): a list of relevant part-of-speeches for elements and features. In case this parameter is not defined all pos are considered relevant. The name of the pos should be defined in capital letters, according to the enum strings of CanonicalPosTag.
* min-count: minimal number of counts for extracted element.

Required modules:

* + Logging
  + text-units-storage-device (input of extracted text units)
  + co-occurrences-storage-device (input of extracted co-occurrences)
  + Optional: prev-elements-storage-device. In case we want the ids that are assigned to the extracted elements to fit the ids that are defined in the given (prev) elements storage.
  + Optional: prev-features-storage-device. In case we want the ids that are assigned to the extracted features to fit the ids that are defined in the given (prev) features storage.
  + The data structure configuration suite.
  + elements-storage-device (output of extracted elements)
  + features-storage-device (output of extracted features)
  + element-feature-counts-storage-device (output of element-feature counts)
  + feature-elements-storage-device (output of feature element lists)

**Module: element-feature-scorer**

Defines the scoring process of element and feature scoring from a given storage of element-feature countings.

Features:

* thread-num: the number of concurrent threads for the scoring process
* feature-scoring-class: the name of the class which implements the eu.excitementproject.eop.distsim.scoring.feature.FeatureScoring interface.

Current options:

* + eu.excitementproject.eop.distsim.scoring.feature.PMI
  + eu.excitementproject.eop.distsim.scoring.feature.Count
  + eu.excitementproject.eop.distsim.scoring.feature.Dice
  + eu.excitementproject.eop.distsim.scoring.feature.ElementConditionedFeatureProb
  + eu.excitementproject.eop.distsim.scoring.feature.TFIDF
* element-scoring-class: the name of the class which implements the eu.excitementproject.eop.distsim.scoring.element.ElementScoring interface.

Current options:

* + eu.excitementproject.eop.distsim.scoring.element.Const
  + eu.excitementproject.eop.distsim.scoring.element.L1Norm
  + eu.excitementproject.eop.distsim.scoring.element.L2Norm

Required modules:

* + Logging
  + The data structure configuration suite.
  + elements-storage-device (input of elements)
  + features-storage-device (input of features)
  + element-feature-counts-storage-device (input of element-feature counts)
  + feature-elements-storage-device (input of feature element lists)
  + element-feature-scores-storage-device (output of element-feature scoring)
  + element-scores-storage-device (output of element scoring)
  + Optional: VectorTruncate

**Module: element-similarity-calculator**

Defines the scoring process of element similarity calculation from a given storage of element-feature scorings.

Features:

* thread-num: the number of concurrent threads for the scoring process
* class: the name of the class which implements the eu.excitementproject.eop.distsim.builders.similarity.ElementSimilarityCalculator interface

Current options:

* + eu.excitementproject.eop.distsim.builders.similarity.GeneralElementSimilarityCalculator
* similarity-scoring-class: the name of the class which implements the eu.excitementproject.eop.distsim.scoring.similarity.ElementSimilarityScoring interface.

Current options:

* + eu.excitementproject.eop.distsim.scoring.similarity.Lin
  + eu.excitementproject.eop.distsim.scoring.similarity.APinc
  + eu.excitementproject.eop.distsim.scoring.similarity.Cosine
  + eu.excitementproject.eop.distsim.scoring.similarity.Cover

Required modules:

* + Logging
  + The data structure configuration suite.
  + feature-elements-storage-device (input of feature elements lists)
  + element-feature-scores-storage-device (input of element-feature scoring)
  + element-scores-storage-device (input of element scoring)
  + elements-similarities-l2r-storage-device (output of l2r element similarities)
  + elements-similarities-r2l-storage-device (output of r2l element similarities)
  + Optional: VectorTruncate

**Module: element-similarity-combiner**

Defines the process of combining several element similarity scorings into one unified score.

Features:

* class: the name of the class which implements the eu.excitementproject.eop.distsim.builders.similarity.ElementSimilarityCombiner interface

Current options:

* + eu.excitementproject.eop.distsim.builders.similarity.OrderedBasedElementSimilarityCombiner
* in-files: a list files which are persistence devices (of type File) of various similarity storages to be combined.
* out-combined-file: the name of the output file, composed of the unified scores
* storage-device-class: the specific type of the storage device for the input and output devices, usually the eu.excitementproject.eop.distsim.storage.File class, or one of its subclasses.
* similarity-combination-class: a name of a class which implements the eu.excitementproject.eop.distsim.scoring.combine.ElementSimilarityCombination interface.

Current options:

* + eu.excitementproject.eop.distsim.scoring.combine.GeometricMean
* is-sorted: are the given in-files sorted (by the id of the elements)? [default – no]
* tmp-dir: in case the files are not sorted, the path of the tmp directory for the Linux 'sort' system call [default – the tmp directory of Linux, usually /tmp/]

**Module: file-to-redis**

Defines the process of converting a general eu.excitementproject.eop.distsim.storage.File device to eu.excitementproject.eop.distsim.storage.Redis

Features:

* class: the specific type of the eu.excitementproject.eop.distsim.storage.File input (can be one of its subclasses).
* file: the path of the similarity input file
* elements-file: the path to the elements file
* redis-host: the host of the output redis server
* redis-port: the port of the redis server

**Module: knowledge-resource**

Defines the parameters of Redis-based knowledge resource

Features:

* resource-name: the name of the resource (as defined in LexicalResource and SyntacticResource interfaces).
* top-n-rules: indicate the number of top rules to be retrieved
* l2r-redis-host: the host of Redis server which contains the left-2-right rules
* l2r-redis-port: the port of Redis server which contains the left-2-right rules
* r2l-redis-host: the host of Redis server which contains the right-2-left rules
* r2l-redis-port: the port of Redis server which contains the right-2-left rules

# Application

The build-model script builds a distributional similarity for a given directory of configuration files.

> build-model <configuration directory>

The script includes following main programs:

* eu.excitementproject.eop.distsim.builders.cooccurrence.GeneralCooccurrenceExtractor <configuration file>

Gets a corpus and generates co-occurrences database, composed of a text-units storage, where each text-unit has a unique id and count, and a co-occurrence storage, composed of two text-unit ids and their relations.

* eu.excitementproject.eop.distsim.builders.elementfeature.GeneralElementFeatureExtractor <configuration file>

Gets a database of co-occurrences, and generates a database of elements and features with their counts, composed of elements storage (where each element is assigned to a unique id and count), feature storage (where each feature is assigned to a unique id and count), element-features storage where each element id is assigned to a list of feature ids with their joint counts, and a feature-elements storage, where each feature id is assigned to a set of elements.

* eu.excitementproject.eop.distsim.builders.elementfeature. ExtractAndCountBasicNodeBasedElementsFeatures <configuration file>

Gets a corpus, and generates a database of elements and features with their counts, composed of elements storage (where each element is assigned to a unique id and count), feature storage (where each feature is assigned to a unique id and count), element-features storage where each element id is assigned to a list of feature ids with their joint counts, and a feature-elements storage, where each feature id is assigned to a set of elements.

This program can be used, instead of the two above programs. In contrast to the above program it does not requires memory, since it is based on the map-reduce scheme.

* eu.excitementproject.eop.distsim.builders.scoring.GeneralElementFeatureScorer <configuration file>

Gets a database of elements and features with their counts, and generates a database of elements and features with their scores, composed of storage of element-feature scores, and storage of element scores.

* eu.excitementproject.eop.distsim.builders.similarity.GeneralElementSimilarityCalculator <configuration file>

Gets a database of elements and features scores, and generates a database of element similarities, where each element id is assigned to a list of entailing element ids with their similarity scores.

* eu.excitementproject.eop.distsim.builders.similarity.GeneralElementSimilarityCombiner <configuration file>

Gets a list of similarity databases, and generates a combined similarity database.

* eu.excitementproject.eop.distsim.storage.File2Redis <configuration file>

Converts a given general storage device of type eu.excitementproject.eop.distsim.storage.File to eu.excitementproject.eop.distsim.storage.Redis

* eu.excitementproject.eop.distsim.storage.ElementFile2Redis <configuration file>

Converts a given storage device of elements of type eu.excitementproject.eop.distsim.storage.File to eu.excitementproject.eop.distsim.storage.Redis

# References

Jonathan Berant, Ido Dagan and Jacob Goldberger

Global Learning of Focused Entailment Graphs, Proceedings of ACL, 2010.

Jonathan Berant, Ido Dagan, Meni Adler and Jacob Goldberger, Efficient Tree-based Approximation for Entailment Graph Learning, ACL, 2012.

Timothy Chklovski and Patrick Pantel. Verbocean: Mining the web

for fine-grained semantic verb relations. In Proceedings of EMNLP, 2004.

Dagan, Ido. Contextual Word Similarity, in Rob Dale, Hermann Moisl and Harold Somers (Eds.), Handbook of Natural Language Processing, Marcel Dekker Inc, 2000, Chapter 19, pp. 459-476.

Kenneth Ward Church and Patrick Hanks, Word Association Norms, Mutual Information, and Lexicography, Computational Linguistics, 16(1):22-29, 1990.

Christiane Fellbaum, ed. WordNet - An Electronic Lexical Database.

The MIT Press, 1998.

Nizar Habash and Bonnie Dorr, A categorial variation database for

english. In Proceedings of NAACL, 2003.

Zellig Harris. Distributional structure. *Word*, 10(23):146-162, 1954.

Lili Kotlerman, Ido Dagan, Idan Szpektor and Maayan Zhitomirsky-Geffet. Directional Distributional Similarity for Lexical Inference. Special Issue of Natural Language Engineering on Distributional Lexical Semantics (JNLE-DLS), 2010.

Lili Kotlerman, Ido Dagan, Idan Szpektor and Maayan Zhitomirsky-Geffet. Directional Distributional Similarity for Lexical Expansion. In Proceedings of ACL (short papers), 2009.

Dekang Lin, Automatic retrieval and clustering of similar words, ACL-COLING, 1998.

Dekang Lin, Dependency-based evaluation of minipar. In Proceedings of the Workshop on Evaluation of Parsing Systems at LREC 1998, Granada, Spain, 1998.

Dekang Lin and Patrick Pantel, Discovery of inference rules for question answering, *Natural Language Engineering*, 7(4):343-360, 2001.

Amnon Lotan, A Syntax-based Rule-base for Textual Entailment and a Semantic Truth Value Annotator, Msc. Thesis, Bar Ilan University, 2012.

W. Lowe, Towards a theory of semantic space' in J. D. Moore and K. Stenning (Eds.) Proceedings of the Twenty-first Annual Meeting of the Cognitive Science Society LEA pp.576-581, 2001.

Shachar Mirkin, Ido Dagan, and Eyal Shnarch. Evaluating the inferential utility of

lexical-semantic resources. In Proceedings of EACL, 2009.

Eyal Shnarch, Libby Barak, Ido Dagan. Extracting Lexical Reference Rules from Wikipedia, ACL, 2009.

K. Sparck Jones, A statistical interpretation of term specificity and its application in retrieval, Journal of Documentation, 28 (1), 1972.

Rion Snow, Daniel Jurafsky, and Andrew Y Ng., Learning syntactic patterns for automatic hypernym discovery. NIPS 17, 2005

Idan Szpektor and Ido Dagan, Learning entailment rules for unary templates, In

Proceedings of COLING, 2008

Julia Weeds and David Weir, A general framework for distributional similarity, In Proceedings of EMNLP , 2003.

Julia Weeds and David Weir, Co-occurrence Retrieval: A Flexible Framework for Lexical Distributional Similarity, *Computational Linguistics*, 31(4): 439-475, 2005.

Hila Weisman, Jonathan Berant, Idan Szpektor and Ido Dagan, Learning Verb Inference Rules from Linguistically-Motivated Evidence, EMNMLP, 2012.