University of Tübingen
Department of General and Computational Linguistics
Wilhelmstr. 19, 72074 Tübingen, Germany

Tutorial:

Semantic Relatedness API for GermaNet

Version 7.0 (November 2012)

Contributions from (alphabetical order):
Agnia Barsukova, Anne Brock, Verena Henrich, Christina Hoppermann, Klaus Suttner, Stefanie Wolf.

Contents

1 Introduction

This manual gives an overview of the semantic relatedness API for GermaNet as well as its GUI extension, providing access to semantic relatedness measures calculated by various algorithms.

If you plan to use the API as a stand-alone application, we suggest you use the GUI version. If you wish to incorporate it as a library into your program, then you should take a look at the `Demo` class in the package `de.tuebingen.uni.sfs.germanet.relatedness`, which illustrates how to use the library.

1.1 Preliminaries

If you have not done so already, you will need to:

1. Obtain[1] and unpack the XML GermaNet data (version 7.0, previous versions are incompatible with the API).

2. Download[2] and unpack the *GermaNetSemRelApi7.0.zip* archive, containing, among other files, *GermaNetSemRelApi7.0.jar*[3] and *relatedness.ini* file, which need to be kept in one folder.

1.2 Javadoc Documentation

The semantic relatedness API and its GUI extension are both contained in two different packages – `de.tuebingen.uni.sfs.germanet.relatedness` and `de.tuebingen.uni.sfs.germanet.relatedness.gui`, respectively. All classes in those packages are documented in the enclosed javadoc documentation which is also available online[4].

1.3 Important Note on Memory

Loading GermaNet requires more memory than the JVM allocates by default, so the application will most likely need to be run with JVM options that increase the memory allocated, such as:

[1] `http://www.sfs.uni-tuebingen.de/GermaNet/licenses.shtml`

[2] `http://www.sfs.uni-tuebingen.de/GermaNet/tools.shtml`

[3] This package contains the third-party stemmer Snowball, distributed under BSD license, the third-party visualization toolkit Prefuse (BSD license), and the GermaNetAPI, version 7 (GPL license).

[4] `http://www.sfs.uni-tuebingen.de/GermaNet/documents/api/javadocSemRel7.0/`

```
java -Xms512m -Xmx512m -jar GermaNetSemRelApi7.0.jar <GERMANET-FOLDER>
                                                     <FREQUENCIES-FOLDER>
```

These options can be added to your IDE's VM options so that they will be used automatically when your application is run from within the IDE. Depending on the memory needs of the application itself, the 512s may need to be changed to a higher number. Be careful not to allocate too much memory to the JVM, though, as this may cause other running programs (like your windowing environment) to crash.

1.4 Structure of this Tutorial

Section 2 briefly introduces all semantic relatedness measures that are implemented in *GermaNetSemRelApi*. Before you can actually start to use either the semantic relatedness API itself or its GUI extension, you will need to know how to calculate the required frequency lists – see Section 3. The graphical user interface is described in Section 4. Finally, Section 5 explains how the semantic relatedness API can be used programmatically.

2 Semantic Relatedness Measures

All methods that are implemented in this semantic relatedness API are briefly described in the following. Please see [1] or [2] for more detailed descriptions of the measures themselves. Note that all these measures are implemented in the `Relatedness` class of the `de.tuebingen.uni.sfs.germanet.relatedness` package. That is, all methods printed in `courier` font refer to actual methods in that class.

1. Path:

   `path(Synset s1, Synset s2)`

   Relatedness between two concepts is computed as a function of the distance between the two corresponding synset nodes s_1 and s_2 and the longest possible 'shortest path' between any two nodes in GermaNet.

   $$sim(s_1, s_2) = \frac{MAXSHORTESTPATH - length(s_1, s_2)}{MAXSHORTESTPATH}$$

   where

   - *length*(s_1, s_2) = shortest path between synsets s_1 and s_2
   - MAXSHORTESTPATH = maximum of all shortest paths.

2. Wu and Palmer [9]:

   `wuAndPalmer(Synset s1, Synset s2)`

   Conceptual similarity between two concepts is computed as the shortest path length (limited to hypernymy/hyponymy relations) between them, normalized by the depth of their lowest common subsumer (LCS).

   $sim(s_1, s_2) =$

   $$= \frac{2 \cdot depth(LCS(s_1, s_2))}{length(s_1, LCS(s_1, s_2)) + length(s_2, LCS(s_1, s_2)) + 2 \cdot depth(LCS(s_1, s_2))}$$

   where

   - *depth(s)* = length of the shortest path from synset s to GNROOT
   - *LCS(s_1, s_2)* = lowest common subsumer of synsets s_1 and s_2.

3. Leacock and Chodorow [5]:

   `leacockAndChodorow(Synset s1, Synset s2)`

   Similarity between two concepts is computed as the negative logarithm of the length of the shortest path between them (limited to hypernymy/hyponymy relations only) over the path length of the overall depth of GermaNet.

   $$sim(s_1, s_2) = -log \frac{length(s_1, s_2)}{2 \cdot MAXDEPTH}$$

   where

   - $MAXDEPTH$ = maximal distance of all synsets s from GNROOT.

4. Resnik [8]:

   `resnik(Synset s1, Synset s2, HashMap<String, Integer> freq)`

   Similarity between two concepts is computed as the information content (IC) of their LCS in the graph. IC of a concept in the GermaNet graph is estimated by the relative frequency of this word in a large corpus.

   $$sim(s_1, s_2) = -log(IC(LCS(s_1, s_2)))$$

   with

   $$IC(s) = \frac{\sum_{w \in W(s)} freq(w)}{N}$$

   where

- $W(s)$ = the set of words w in synset s
- N = total number of words in the specified corpus.

5. Lin [7]:

   `lin(Synset s1, Synset s2, HashMap<String, Integer> freq)`

   Similarity between two concepts is measured as the IC (multiplied by two) of their LCS over the sum of the ICs of the concepts.

$$sim(s_1, s_2) = \frac{2 \cdot log(IC(LCS(s_1, s_2)))}{log(IC(s_1)) + log(IC(s_2))}$$

6. Jiang and Conrath [4]:

   `jiangAndConrath(Synset s1, Synset s2, HashMap<String, Integer> freq)`

   Similarity between two concepts is computed as the inverse of their distance (measured as the sum of the ICs of the concepts minus double the IC of their LCS).

$$sim(s_1, s_2) = 2 \cdot log(IC(LCS(s_1, s_2))) - log(IC(s_1)) - log(IC(s_2))$$

7. Hirst and St-Onge [3]:

   `hirstAndStOnge(Synset s1, Synset s2)`
   `hirstAndStOnge(Synset s1, Synset s2, double C, double k)`

   For computing the semantic relatedness between two concepts, the length of the shortest path between the concepts (not limited to the hypernymy/hyponymy relations) and the change of "direction" (i.e., the relations in GermaNet can be grouped into upwards, downwards, and horizontal) are considered.

$$sim(s_1, s_2) = C - length(s_1, s_2) - k \cdot d$$

   where

   - C = maximum value for medium strength relations (preset to 10)
   - k = factor for the number of direction changes (preset to 1)
   - d = the number of changes of direction in the shortest path.

8. Adapted Lesk [6]:

   ```
   lesk(Synset s1, Synset s2, GermaNet gnet)
   lesk(Synset s1, Synset s2, GermaNet gnet,
   SnowballStemmer stemmer, int size, int limit, boolean
   oneOrthForm, boolean hypernymsOnly, boolean
   includeGermanetGloss, boolean includeWiktionaryGlosses)
   ```

   Lexical fields, i.e., bags of words from surrounding synsets, are examined for calculating word overlaps. Optionally, paraphrases from GermaNet and Wiktionary can also contribute to this calculation.

   $$sim(s_1, s_2) = lexField(s_1) \cap lexField(s_2)$$

   where

   - $lexField(s_i)$ = all words $w \in s_i$ with $length(s_i, s_j) \leq size$ and $length(s_j, GNROOT) > limit$, by default $size = 4$ and $limit = 2$
   - $oneOrthForm$ = true if only one orthForm of each synset will be considered, false if all of them are considered (default false)
   - $hypernymsOnly$ = true if only the hypernymy relation is used; otherwise, all relations except hyponymy are used (default true)
   - $includeGermanetGloss$ and $includeWiktionaryGlosses$ = true if those are to be included into the lexical field (default false).

3 Frequency Lists

Some of the relatedness measures require word frequency data, therefore you need to provide it on launching the program. The frequency file(s) should be in the format of list(s) with tab-separated *frequency – word* pairs. You need to have a separate file for each part of speech. Note that the API is searching the filenames for "adj", "noun", and "verb", i.e., it is necessary that you include "adj", "noun", and "verb" into the respective filenames.

When you use the semantic relatedness API programmatically, you will need to call the static method `assignFrequencies` in the `Frequency` class of the package `de.tuebingen.uni.sfs.germanet.relatedness` in order to convert your data into the appropriate format of *synsetID – frequency* pairs.

To improve the mapping coverage between your word lists and the GermaNet synsets, you first might want to clean your lists (remove non-word characters absent from GermaNet, normalize word entries, etc.) with the static method `cleanLists`.

The `Demo` class in `de.tuebingen.uni.sfs.germanet.relatedness`, meant to illustrate the package functionality, calls both methods:

```
import de.tuebingen.uni.sfs.germanet.api.*;
import de.tuebingen.uni.sfs.germanet.relatedness.*;
import java.io.*;
import java.util.*;

public class Demo {
    public static void main(String[] args) {

        if (args.length < 3) {
            System.out.println("USAGE: java Main outputFile," +
             "pathToGermaNet, frequencyDirectory");
            System.exit(0);
        }
        try {
            ...
            GermaNet gnet = new GermaNet(args[1], true); //ignore case
            ...
            Frequency.cleanLists(args[2]);
            Frequency.assignFrequencies(args[2], gnet);
            ...
            HashMap<String,Integer> frequencies =
                    Frequency.loadFreq(args[2]+"/frequencies.csv");
        }
    }
}
```

The *GermaNetSemRelApi* GUI application automatically calls the `assignFrequencies` method to create a converted frequency file if it does not yet exist (but it will not clean your source files).

Please note that the methods for computing and formatting frequency lists can take up a lot of time, but in general they will only need to be called once, taking no additional time in subsequent launches.

4 Using the GUI

To start the graphical user interface for calculating semantic relatedness measures, you will need to launch *GermaNetSemRelApi7.0.jar* with two parameters: (i) The path to the directory containing the GermaNet data (<GERMANET-FOLDER>) and (ii) the path to the directory containing the frequency list(s)[5] (<FREQUENCIES-FOLDER>):

```
java -jar GermaNetSemRelApi7.0.jar <GERMANET-FOLDER> <FREQUENCIES-FOLDER>
```

Note that the enclosed file *relatedness.ini* has to be in the same directory as the .jar file itself. This call will start the `MainFrame` class in package `de.tuebingen.uni.sfs.germanet.relatedness.gui`.

[5]Section 3 will gives more information on the frequency list(s).

4.1 Calculating Relatedness

Once you have started the GUI, you can enter the words between which you want to calculate semantic relatedness into the corresponding text fields *Word 1* and *Word 2*, and then choose the synsets you are interested in from the drop-down lists on the right – see Figure 1. To illustrate which sense each synset refers to, you are provided with a list of all corresponding lexical units, as well as word class, hypernym, and paraphrase information displayed in tooltips.

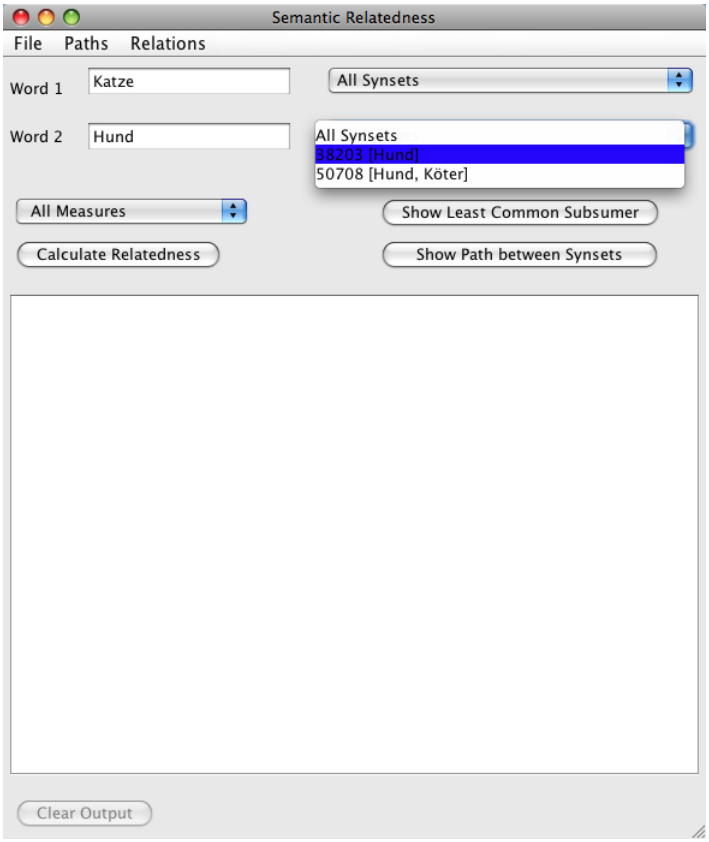

Figure 1: Graphical user interface to the semantic relatedness API

You can either select a specific semantic relatedness measure you are interested in from a drop-down list or just use the default option *All Measures* to calculate all provided measures at once.

Another way of providing input for the GUI version is to upload a tab-separated list of word pairs – each pair written on a separate line, – by choosing *Open File with the word list...* in the File menu. In this case, the

program will calculate results for all matching synsets.

4.2 Paths in GermaNet

Apart from calculating semantic relatedness measures, the GUI application can display paths between synsets in GermaNet. Currently, this functionality is limited to the hypernymy relation of the synsets. Paths are displayed in new windows, which you can manipulate in the following way: you can zoom in and out by using the scrolling wheel, and drag the path visualization by holding the left mouse button and moving the mouse.

Shortest Path between Synsets

Using the *Show Path between Synsets* button, you can visualize the shortest path between two synsets. The function takes the words entered into the text fields *Word 1* and *Word 2* as input. If you have not yet picked a single synset corresponding to the given word, a dialogue menu will ask you to do so. The shortest path(s) will then be displayed in a new window.

Paths to the GermaNet Root

With the help of the *Path* menu, you can visualize either the shortest or all paths to the GermaNet root from a particular synset. The path(s) will be displayed in separate window(s) and printed to the textbox. Another option in the *Path* menu is *Distance to Root*, which will print the corresponding numerical value to the textbox.

4.3 Incorporating GermaNet Relations

Another functionality of the GUI is that using the *Relations* menu, you can access synsets linked to the one you provide by the relations of synonymy, hypernymy/hyponymy, or meronymy/holonymy available within GermaNet. The resulting synsets are written to the textbox in the main GUI window.

5 Programmatic Use of the API

Add *GermaNetSemRelApi7.0.jar* to the classpath of your Java application. If you are working with an IDE (such as NetBeans or Eclipse), add *GermaNet-SemRelApi7.0.jar* to the classpath for any project which uses the semantic

relatedness API for GermaNet.[6]

Before you can start using the semantic relatedness API programmatically, you need to know how to create `GermaNet` and `Synset` objects that are defined in the GermaNet Java API [7]. To get started with using the semantic relatedness API programmatically, the `Demo` class included in the package `de.tuebingen.uni.sfs.germanet.relatedness` gives you an example of how to calculate semantic similarity. In order to calculate relatedness measures, you need to specify which two synsets you need them for. The *Demo* class will prompt you for two words, and then will proceed to pick the first synsets from the lists of matches for each word.

Basically, all relatedness measures described in Section 2 are available in the `Relatedness` class in package `de.tuebingen.uni.sfs.germanet.relatedness`. That is, you first need to create an object of the `Relatedness` class, providing a `GermaNet` object as an argument:

```
Relatedness rel = new Relatedness(gnet);
```

When calling one of the methods of the `Relatedness` class to calculate a semantic relatedness measure, you need to provide two `Synset` objects. Note that all methods printed in `courier` font in Section 2 refer to actual methods in the `Relatedness` class. Please take a look at the provided `Demo` class to see an example usage of all these semantic relatedness measures. The input and output data format for the methods in the `Relatedness` class is documented in the enclosed javadoc documentation which is also available online[8].

[6] See `http://faq.javaranch.com/java/HowToSetTheClasspath` for help with setting your classpath on various operating systems.

[7] Please see the documentation of the GermaNet Java API (`http://www.sfs.uni-tuebingen.de/GermaNet/tools.shtml#APIs`) in order to learn how to create `GermaNet` and `Synset` objects.

[8] `http://www.sfs.uni-tuebingen.de/GermaNet/documents/api/javadocSemRel7.0/`

Bibliography

[1] A. Budanitsky and G. Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and other lexical resources, second meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[2] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. In *Computational Linguistics, 32(1)*, pages 13–47, 2006.

[3] G. Hirst and D. St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In C. Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 305–332, 1998.

[4] J. Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the International Conference on Research in Computational Linguistics, ROCLING X*, pages 19–33, 1997.

[5] C. Leacock and M. Chodorow. Combining local context and wordnet similarity for word sense identification. In C. Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 305–332, 1998.

[6] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation, SIGDOC '86*, pages 24–26, 1986.

[7] D. Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98*, pages 296–304, 1998.

[8] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th international joint conference on Artificial intelligence, volume 1*, pages 448–453, 1995.

[9] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics, ACL '94*, pages 133–138, 1994.