Introduction to GermaNet and Word Sense Disambiguation

Programming Course: Computational Linguistics I – Verena Henrich
Objectives

• Learn the basics of GermaNet and its structure: synsets, lexical units, and relations between them
• Get an idea of what word sense disambiguation (WSD) is and how this is connected to GermaNet
• Introduce Lesk, a very simple WSD algorithm
• Explore a way of how to access GermaNet programmatically with an API
GermaNet: A German Wordnet

• GermaNet is a lexical semantic network covering the German base vocabulary

• Belongs to the family of wordnets modeled after the Princeton WordNet for English

• GermaNet is divided into 3 word categories:
  - Adjectives
  - Nouns
  - Verbs

• Words are ordered according to their meaning
Lexical Units in GermaNet

• Word meanings are represented by *lexical units*

• A lexical unit specifies one form and one meaning (i.e. reading) of a word

• Examples:

  - “Bank“ has 2 readings
    - Reading 1: `[Bank, {Sitzbank}]` *(bench)*
    - Reading 2: `[Bank, {Geldinstitut}]` *(financial institution)*

  - “Leiter“ has 3 readings
    - Reading 1: `[Leiter, {Steiggerät}]` *(ladder)*
    - Reading 2: `[Leiter, {Verantwortlicher, Anführer}]` *(leader)*
    - Reading 3: `[Leiter, {stromleitender Stoff}]` *(electric conductor)*

• Lexical units are grouped into semantic concepts according to their meaning
Synsets in GermaNet

- Semantic concepts are represented by synsets.
- A synset is a set of (near-)synonymous words.
Lexical Relations in GermaNet

- Lexical relations hold between two lexical units
  - Synonymy
  - Antonymy
  - Pertainymy
Conceptual Relations in GermaNet

- Conceptual relations hold between synsets
  - Hypernymy and hyponymy
  - Part-whole relations
  - Entailment
  - Causation
  - Association
Part-Whole Relations in GermaNet

• 4 kinds of part-whole relations
  - Component meronymy
  - Portion meronymy
  - Substance meronymy
  - Member meronymy
Part-Whole Relations in GermaNet

Schiff  Member Meronymy  Flotte

Schnee  Substance Meronymy  Schneemann
Lexical and Conceptual Relations
Readings for “unterhalten”

1. (v) [unterhalten, pflegen] (to cultivate) -- über etwas verfügen
   • [pflegen]
   Hypernyms: [haben, besitzen]

2. (v) [unterhalten] (to keep oneself amused) -- sich auf angenehme Weise die Zeit vertreiben
   • [unterhalten] -- NN.AR.BM -- Sie hat sich blendend unterhalten. (NN.AR.BM)
   Hypernyms: [vergnügen]

3. (v) [unterhalten] (to entertain) -- für Zerstreuung/Zeitvertreib sorgen
   • [unterhalten] -- NN.AN.Bs -- Er unterhielt seine Gäste mit Musik. (NN.AN.Bs)
   Hypernyms: [vergnügen, amüsieren]

4. (v) [unterhalten] (to maintain sth.) -- etw. halten/einrichten/betreiben und dafür aufkommen
   • [unterhalten] -- NN.AN -- Er unterhält einen Reitstall. (NN.AN)
   Hypernyms: [führen]
   Hyponyms: [instandhalten] [bewirtschaften]

5. (v) [unterhalten] (to talk) -- ein Gespräch führen
     -- Er unterhielt sich nur mit mir. (NN.AR.Bo)
   Hypernyms: [austauschen]
   Hyponyms: [klönen] [labern] [palavern] [philosophieren] [plaudern] [plaudern, schwatzen, schnattern]

6. (v) [unterhalten, alimentieren] (to support sb.) -- für jmds. Lebensunterhalt aufkommen
   • [unterhalten] -- NN.AN -- Er unterhält eine sieben-köpfige Familie. (NN.AN)
   • [alimentieren]
   Hypernyms: [ernähren, nähen]
Web Application for GermaNet

http://weblicht.sfs.uni-tuebingen.de/germanet/GernWeb6.0/
Introduction to Word Sense Disambiguation (WSD)

Reading:
Word Sense Disambiguation: Algorithms and Applications
by Eneko Agirre and Philip Glenny Edmonds, chapter 5.2
http://books.google.de/books?id=GLck75U20pAC&pg=PA109
The Purpose of GermaNet

• GermaNet is primarily intended to serve as a resource for word sense disambiguation which is crucial for natural language applications like
  - Information retrieval
  - Construction of language technology tools
  - Annotation of corpora
  - Machine translation
What is Word Sense Disambiguation (WSD)?

• Words are ambiguous

• Which sense of bank is meant here? (financial institution or riverbank)

  The bank profits from the difference between the level of interest it pays for deposits, and the level of interest it charges in its lending activities.

• The task of WSD is to identify the correct sense of a word in a specific context – assuming a fixed inventory of word senses
Lesk Algorithm

• The Lesk algorithm (Lesk 1986) is one of the first algorithms for the semantic disambiguation of all words

• Required resource: a dictionary with definitions for each possible word sense

• Idea: disambiguate words by computing the word overlap among their sense definitions, i.e., word sense whose definition has maximum overlap with the definitions of each of the context words is chosen
Lesk Algorithm:
Pseudocode for disambiguating two words

• for each sense $i$ of word $W_1$
  - for each sense $j$ of word $W_2$
    ▪ compute $\text{overlap}(i,j)$, the number of (non-stop) words in common between the definition of sense $i$ and sense $j$
  
• find $i$ and $j$ for which $\text{overlap}(i,j)$ is maximized

• assign sense $i$ to $W_1$ and $j$ to $W_2$
Lesk Algorithm – Example

• Example context: ...pine cone...

2 word senses of pine

#1 kinds of evergreen tree with needle-shaped leaves

#2 waste away through sorrow or illness

3 word senses of cone

#1 solid body which narrows to a point

#2 something of this shape whether solid or hollow

#3 fruit of a certain evergreen tree

Sense inventory contains 2 word senses for pine and 3 word senses for cone
Lesk Algorithm – Example

• Example context:  ...pine cone...

<table>
<thead>
<tr>
<th>2 word senses of <em>pine</em></th>
<th>3 word senses of <em>cone</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 kinds of <strong>evergreen tree</strong> with needle-shaped leaves</td>
<td>#1 solid body which narrows to a point</td>
</tr>
<tr>
<td>#2 waste away through sorrow or illness</td>
<td>#2 something of this shape whether solid or hollow</td>
</tr>
<tr>
<td></td>
<td>#3 fruit of a certain <strong>evergreen tree</strong></td>
</tr>
</tbody>
</table>

\[
\text{overlap}(\text{pine}\#1, \text{cone}\#1) = \text{overlap}(\text{pine}\#2, \text{cone}\#1) = \text{overlap}(\text{pine}\#1, \text{cone}\#2) = \text{overlap}(\text{pine}\#2, \text{cone}\#2) = \text{overlap}(\text{pine}\#2, \text{cone}\#3) = 0
\]

\[
\text{overlap}(\text{pine}\#1, \text{cone}\#3) = \{\text{evergreen} + \text{tree}\} = 2
\]

→ pine#1 and cone#3 are selected
Problem with the Original Lesk Algorithm

• Problem: Combinatorial explosion of possible word sense combinations when more than two words are considered

• Example: "I saw a man who is 98 years old and can still walk and tell jokes."
  - Several possible senses for ambiguous words: see(26), man(11), year(4), old(8), can(5), still(4), walk(10), tell(8), joke(3)
  - 43,929,600 sense combinations

• Solution: simplified version of the Lesk algorithm
Simplified Lesk Algorithm

• Tries to solve the combinatorial explosion of word sense combinations by disambiguating each ambiguous word in the input text individually, regardless of the meaning of the other words occurring in the same context.

• Correct meaning of each word is determined by finding the sense that leads to the highest overlap between its dictionary definition and the current context.

• Simplified Lesk can significantly outperform the original algorithm in terms of precision and efficiency.
Simplified Lesk Algorithm: Pseudocode to disambiguate a target word

• for each sense $i$ of $W$
  
  - determine $\text{overlap}(i, c)$, the number of (non-stop) words in common between the definition of sense $i$ and current sentential context $c$

• find sense $i$ for which $\text{overlap}(i, c)$ is maximized

• assign sense $i$ to $W$
Simplified Lesk Algorithm – Example

<table>
<thead>
<tr>
<th>2 word senses of bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 sloping land (especially the slope beside a body of water)</td>
</tr>
<tr>
<td>#2 a financial institution that accepts deposits and channels the money into lending activities</td>
</tr>
</tbody>
</table>

Target word: bank

Example context: The bank profits from the difference between the level of interest it pays for deposits, and the level of interest it charges in its lending activities.
Simplified Lesk Algorithm – Example

Example context: *The* bank* profits from the difference between the level of interest it pays for deposits, and the level of interest it charges in its lending activities.*

<table>
<thead>
<tr>
<th>2 word senses of <em>bank</em></th>
<th>overlap(bank#1, context) = 0</th>
<th>overlap(bank#2, context) = {deposits + lending + activities} = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 sloping land (especially the slope beside a body of water)</td>
<td></td>
<td>→ bank#2 is selected</td>
</tr>
</tbody>
</table>
| #2 a financial institution that accepts deposits and channels the money into lending activities |                                           | }
Simplified Lesk Algorithm for GermaNet

• Problem: GermaNet does not contain comprehensive sense definitions for all word senses

• Solution:
  - Represent a word sense by a *bag of words* (BOW) that includes all synonyms and related words
  - Calculate overlap by counting the words in common between the BOW of a word sense and the current sentential context

• BOW: can contain all words that are linked to the word sense in question, e.g., all synonyms, antonyms, hypernyms, hyponyms, meronyms, holonyms, etc.
Simplified Lesk Algorithm for GermaNet – Example

Example context: Der Verlag hatte sein eigenes Kapital in eine feste Geldanlage umgewandelt, während die Kohle anderer Unterstützer für laufende Kosten verwendet wurde.

2 word senses of Kohle in GermaNet (‘money’ and ‘coal’)

<table>
<thead>
<tr>
<th>Sense</th>
<th>Synonyms</th>
<th>Hypernyms</th>
<th>Holonyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Geld, Kies, Knete, Moneten</td>
<td>Zahlungsmittel, Finanzmittel</td>
<td>Geldvermögen, Geldmittel, Mittel, Kapital, Finanzausstattung, Geldkapital, Geldanlage, Anlage, Asset</td>
</tr>
<tr>
<td>#2</td>
<td>Kohlegestein</td>
<td>Rückstandsgestein</td>
<td>Importkohle, Rohkohle, Aktivkohle, Torf, Braunkohle, Steinkohle, Koks, Kokskohle, Anthrazit, Grafit, Glanzkohle, Xylit</td>
</tr>
</tbody>
</table>
Simplified Lesk Algorithm for GermaNet – Example

Example context: Der Verlag hatte sein eigenes Kapital in eine feste Geldanlage umgewandelt, während die Kohle anderer Unterstützer für laufende Kosten verwendet wurde.

<table>
<thead>
<tr>
<th>2 word senses of Kohle in GermaNet (‘money’ and ‘coal’)</th>
</tr>
</thead>
<tbody>
<tr>
<td>⬇️ #1 synonyms: Geld, Kies, Knete, Moneten; hypernyms: Zahlungsmittel, Finanzmittel; holonyms: Geldvermögen, Geldmittel, Mittel, Kapital, Finanzausstattung, Geldkapital, Geldanlage, Anlage, Asset</td>
</tr>
<tr>
<td>⬇️ #2 synonyms: Kohlegestein; hypernyms: Rückstandsgestein; hyponyms: Importkohle, Rohkohle, Aktivkohle, Torf, Braunkohle, Steinkohle, Koks, Kokskohle, Anthrazit, Grafit, Glanzkohle, Xylit</td>
</tr>
</tbody>
</table>
Application Programming Interface (API) for GermaNet
Using the GermaNet Java-API

• The Java-API to GermaNet represents a programming interface, i.e., it provides several methods to access GermaNet

• It is intended to be a read-only resource, no methods to extend or modify data are provided

• Before you start, make sure you have access to

  - The GermaNet XML data:  
    http://www.sfs.uni-tuebingen.de/~vhenrich/ss15/java/gn_data/GN_V60.zip

  - The GermaNet Java-API called GermaNetApi6.0.1.jar:  
    http://www.sfs.uni-tuebingen.de/~vhenrich/ss15/java/SelfTests/st4_GermaNet/GermaNetApi6.0.1.jar
Using the GermaNet Java-API

• All classes and methods are described in the Javadoc documentation:
  http://www.sfs.uni-tuebingen.de/GermaNet/javadoc6.0/index.html

• Add GermaNetApi6.0.jar to the classpath of your project

• Increase the allocated memory by using the following virtual machine options:  
  -Xms256m  -Xmx256m
First Step: Creating a GermaNet Object

• Include the germanet library:
  ```java
  import germanet.*;
  ```

• Main class is called `GermaNet` and serves as a starting point

• When `GermaNet` object is constructed, data is loaded from XML files

• The call to the constructor might throw an exception. Remember from last semester: whenever we read files, we need exception handling to prevent the case that the files are not found
First Step: Creating a GermaNet Object

• Constructor loads GermaNet data from a directory which is specified by a parameter

  - Either as a String object:
    ```java
    GermaNet gnet = new GermaNet("GN_V60");
    ```

  - Or as a File object:
    ```java
    File gnetDir = new File("GN_V60");
    GermaNet gnet = new GermaNet(gnetDir);
    ```

• Optional ignoreCase parameter
Getting Synsets from a GermaNet Object

- The **GermaNet** object provides methods for retrieving lists of synsets (class **Synset**) and lexical units (class **LexUnit**), which can be filtered by e.g., word category or orthographic form.

- Retrieve list of all **Synsets**:  
  ```java
  List<Synset> allSynsets = gnet.getSynsets();
  ```

- Retrieve list of all **Synsets** containing “Bank”:
  ```java
  List<Synset> bankSynsets = gnet.getSynsets(“Bank”);
  ```

- Retrieve list of all **Synsets** with word category adjective:
  ```java
  List<Synset> adjSynsets =
  gnet.getSynsets(WordCategory.adj);
  ```
Working with Synsets (Extract Paraphrase and WordCategory)

- A **Synset** object has methods for retrieving its word category, lexical units, and paraphrases, as well as methods for retrieving synsets that are related to it.

- Retrieving the paraphrase:
  ```java
  String paraphrase = aSynset.getParaphrase();
  ```

- To get a **Synset**'s word category and do further processing in case of an adjective:
  ```java
  WordCategory wc = aSynset.getWordCategory();
  if (wc == WordCategoryCategory.adj) {
      // do something
  }
  ```
Working with Synsets (Extract LexUnits and OrthForms)

• To get a list of all *LexUnits* of a *Synset* and iterate through them:

```java
List<LexUnit> lexList = aSynset.getLexUnits();
for (LexUnit lu : lexList) {
    // process lexical unit
}
```

• To get a *Synset*'s orthographic forms (retrieves a list of all orthographic forms in all the *LexUnits* that belong to this *Synset*):

```java
List<String> orthForms = aSynset.getAllOrthForms();
```
Working with Synsets (Extract Related Synsets)

• To extract a list of all `Synsets` that are related to `aSynset`:
  ```java
  List<Synset> allRelations =
      aSynset.getRelatedSynsets();
  ```

• Suppose you want to find all hypernyms of a `Synset`:
  ```java
  List<Synset> hypernyms =
      aSynset.getRelatedSynsets(ConRel.has_hypernym);
  ```

• Suppose you want to find all hyponyms of a `Synset`:
  ```java
  List<Synset> hyponyms =
      aSynset.getRelatedSynsets(ConRel.has_hypernym);
  ```

• See the Javadoc for more values of `ConRel.xxx`
Getting LexUnits from a GermaNet Object

- Works analogously to extracting **Synsets**
- Retrieve list of all **LexUnits**:
  ```java
  List<LexUnit> allLexUnits = gnet.getLexUnits();
  ```
- Retrieve list of all **LexUnits** containing “Bank”:
  ```java
  List<LexUnit> bankLexUnits =
  gnet.getLexUnits(“Bank”);
  ```
- Retrieve list of all **LexUnits** with word category noun:
  ```java
  List<LexUnit> nomLexUnits =
  gnet.getLexUnits(WordCategory.nomen);
  ```
Working with LexUnits
(Extract OrthForm, Check Named Entity & Artificial)

• To retrieve the main orthographic form of a lexical unit:
  
  ```java
  String orthForm = aLexUnit.getOrthForm();
  ```

• To check whether a lexical unit is a named entity:
  
  ```java
  boolean isNamedEntity = aLexUnit.isNamedEntity();
  ```

• To check whether a lexical unit is artificial:
  
  ```java
  boolean isArtificial = aLexUnit.isArtificial();
  ```

• See the Javadoc for a complete list of methods
Working with LexUnits (Extract Related LexUnits)

• To extract a list of all `LexUnits` that are related to `aLexUnit`:
  ```java
  List<LexUnit> allRelations = aLexUnit.getRelatedLexUnit();
  ```

• Suppose you want to find all antonyms of a `LexUnit`:
  ```java
  List<LexUnit> antonyms = aLexUnit.getRelatedLexUnit(LexRel.has_antonym);
  ```

• See the Javadoc for more values of `LexRel.xxx`
Two Ways of Extracting Synonyms – Solution 1

• Suppose you want to find all synonyms of “Medikament” (note that there is only one reading of “Medikament” in GermaNet)

• First solution:
  1. Extract the one `LexUnit` representing “Medikament”:
     ```
     LexUnit medLU = 
     gnet.getLexUnits(“Medikament“).get(0);
     ```
  2. Extract synonyms of `medLU`:
     ```
     List<LexUnit> synonyms = 
     medLU.getRelatedLexUnits(LexRel.has_synonym);
     ```
Two Ways of Extracting Synonyms – Solution 2

• Second solution:
  1. Extract the one `Synset` representing “Medikament”:
     ```java
     Synset medSynset =
     gnet.getSynsets("Medikament").get(0);
     ```
  2. Extract `LexUnits` of `medSynset`:
     ```java
     List<LexUnit> synonyms =
     medSynset.getLexUnits();
     ```

• Note that the list of synonyms differ in the two solutions: The list in solution 1 contains the synonymous `LexUnits` only, whereas the list in solution 2 also contains the `LexUnit` representing “Medikament”.

Two Ways of Extracting Synonyms – Solution 2

Programming Course: Computational Linguistics I

Verena Henrich
Exercise 1: How many readings have “Dollar“, “haben“, and “sauer“, respectively?
Exercise 1: How many readings have “Dollar“, “haben“, and “sauer“, respectively?

```java
List<LexUnit> dollarLUs = gnet.getLexUnits("Dollar");
System.out.println("\"Dollar\": " + dollarLUs.size());
-> "Dollar": 15

List<LexUnit> schauenLUs = gnet.getLexUnits("schauen");
System.out.println("\"schauen\": " + schauenLUs.size());
-> "schauen": 5

List<LexUnit> sauerLUs = gnet.getLexUnits("sauer");
System.out.println("\"sauer\": " + sauerLUs.size());
-> "sauer": 2
```
Exercise 2: Identify 10 Synsets in GermaNet that contain at least 10 synonymous LexUnits
Exercise 2: Identify 10 Synsets in GermaNet that contain at least 10 synonymous LexUnits

```java
List<Synset> atLeast10LexUnits = new ArrayList<Synset>();
for (Synset synset : gnet.getSynsets()) {
    if (atLeast10LexUnits.size() >= 10)
        break;
    if (!atLeast10LexUnits.contains(synset)
        && synset.getLexUnits().size() >= 10) {
        atLeast10LexUnits.add(synset);
    }
}

System.out.println("10 synsets with at least 10 "
    + "synonymous LUs: ");
for (Synset synset : atLeast10LexUnits)
    System.out.println(synset.getAllOrthForms()
        + synset.getId());
```
Exercise 3: Find 8 words in GermaNet that show at least 8 readings
Exercise 3: Find 8 words in GermaNet that show at least 8 readings

```java
List<String> atLeast8Readings = new ArrayList<String>();

for (LexUnit lu : gnet.getLexUnits()) {
    if (atLeast8Readings.size() >= 8) {
        break;
    } else if (!atLeast8Readings.contains(lu.getOrthForm()) && gnet.getLexUnits(lu.getOrthForm()).size() >= 8) {
        atLeast8Readings.add(lu.getOrthForm());
    }
}

System.out.println("8 words with at least 8 readings: " + atLeast8Readings);
```
Exercise 4: Try to find 5 synsets that have exactly 5 hyponyms
Exercise 4: Try to find 5 synsets that have exactly 5 hyponyms

```java
List<Synset> exactly3Hyponyms = new ArrayList<Synset>();
for (Synset synset : gnet.getSynsets()) {
    if (exactly3Hyponyms.size() >= 5)
        break;

    if (!exactly3Hyponyms.contains(synset) &&
    synset.getRelatedSynsets(ConRel.has_hyponym).size() == 5) {
        exactly3Hyponyms.add(synset);
    }
}

System.out.println("3 synsets with exactly 3 hyponyms: ");
for (Synset synset : exactly3Hyponyms)
    System.out.println(synset.getAllOrthForms() + synset.getId());
```
Exercise 5: Extract related Synsets of a given Synset, where the relation type is provided as a String

- Sometimes you may have a conceptual relationship represented as a *String*. 
Exercise 5: Extract related Synsets of a given Synset, where the relation type is provided as a String

- Sometimes you may have a conceptual relationship represented as a `String`. The following code can be used to validate the String and retrieve the relations of `aSynset`:

```java
String aRel = "has_hypernym";
List<Synset> relList;

// make sure aRel is a valid conceptual relation
if (ConRel.isConRel(aRel)) {
    relList = aSynset.getRelatedSynsets(ConRel.valueOf(aRel));
}
```
Exercise 6: Extract all nominal LexUnits, but without named entities and without artificial nouns
Exercise 6: Extract all nominal LexUnits, but without named entities and without artificial nouns

```java
List<LexUnit> lexList =
        gnet.getLexUnits(WordCategory.nomen);
LexUnit aLexUnit;
Iterator<LexUnit> iter = lexList.iterator();
while (iter.hasNext()) {
    aLexUnit = iter.next();
    if (aLexUnit.isNamedEntity() || aLexUnit.isArtificial()) {
        iter.remove();
    }
}
```

- Note that we use a real `Iterator` object here instead of just a simple `for`-loop because it is the only safe way to remove elements from a `List` while iterating.