Natural Language Processing

Semantics – Word Senses

Mats Dahllöf
Institutionen för lingvistik och filologi
December 2014
Semantics in NLP – this and next time

- Word Sense Disambiguation. Introduction to the assignment.
  
  Lexical semantics and NLP.

- Semantics in NLP – a more general overview.
  
  Questions about the assignment.
Lexical ambiguity: polysemy

One graph word – several senses.

Polysemy: One and the same word represents different concepts depending on context (also when language is known). This is typically due to semantic shifts (like metaphor) from an original sense.

Example:
serve as in *serve food*
serve as in *serve as U.S. ambassador to Norway*
serve as in *served time in prison*

Polysemy is due to semantic shifts that are a natural aspect of language use – More or less all common content words develop polysemy, i.e. a very common phenomenon.
Lexical ambiguity: polysemy

One graph word – several senses.

▶ bachelor
  ▶ kind of student
  ▶ kind of academic degree holder
  ▶ unmarried man
  ▶ young male seal

▶ tak (Swedish)
  ▶ innertak (cieling)
  ▶ yttertak (roof)
Lexical ambiguity: homography

- Homography: same graphic form for two lexically/grammatically different word forms,
- e.g. the verb and adjective form *left* or
- the verb and noun forms *bow*, which are homographs, but not homophones.
- Homography is typically due to random identity in graphic form and is less common than polysemy.
- Homographs will often be disambiguated by grammatical tagging and/or parsing.
  
  *We left...* (verb) vs ... *to the left...* (adjective)
Homonymy and polysemy

In one graph word:

- **Homonymy:** *bass* (fish) vs *bass* (with different pronunciation – music/sound).

- **Polysemy:** a word may represent different concepts in different contexts.
  - *bass* (fish)
    - several different species, i.e. different concepts, etc.
  - *bass* (music/sounds)
    - many different bass instruments, i.e. different concepts.
    - a sound “component” in music.
    - a voice type, etc.
WordNet via wordvis.com, "chair"
WordNet via WordVis
Lexical ambiguity – WSD

- Resolving lexical ambiguity – As an NLP task: Word Sense Disambiguation (WSD).

- Useful in applications like:
  - Machine translation
  - Information extraction
  - Text classification
  - Speech synthesis (bass, bow)
Word senses

- Discerning word senses (for a lemma) – lexicographical task, matter of sophisticated linguistic judgements.

- Theoretical principles. Practical purpose.

- Different linguists/dictionaries make different analyses.

Senses of *day* in WordNet, for instance (1)

- (169) S: (n) day (1:28:00::), twenty-four hours (*Sw. dygn*)
- (70) S: (n) day (1:28:03::) (some point or period in time) "it should arrive any day now"; "those were the days"; "these days it is not unusual"
- (54) S: (n) day (1:28:01::) (a day assigned to a particular purpose or observance) "Mother’s Day"
- (38) S: (n) day (1:28:02::), daytime
- (11) S: (n) day (1:28:04::) (the recurring hours when you are not sleeping (especially those when you are working))
Senses of *day* in WordNet, for instance (2)

- (7) S: (n) day (1:28:05::) (an era of existence or influence) ”in the day of the dinosaurs”;
- S: (n) day (1:28:07::) (the period of time taken by a particular planet (e.g. Mars) to make a complete rotation on its axis)
- S: (n) sidereal day (sidereal 1:28:00::), day (1:28:06::) (the time for one complete rotation of the earth relative to a particular star, about 4 minutes shorter than a mean day)
- S: (n) day (1:26:00::) (a period of opportunity) ”he deserves his day in court”; ”every dog has his day”
Word Sense Disambiguation (WSD)

- A distributional hypothesis for WSD: words representing the same sense have more similar distributions than words representing different senses.
  
  I.e. distribution similarity implies sense similarity.

- We can use this for supervised learning of WSD.
  
  This requires data in the form of a sense-tagged corpus (based on a given sense inventory, e.g. the one given by WordNet).
Manual sense-tagging

- More difficult than typical grammatical tagging: Finer categories and more subtle judgements.

- As we saw in the *day* example, senses and their distinctions can be quite sophisticated. Definitions and examples are often far from obvious.

- Expensive: requires competent people and standardised procedures.

- Quality measure: inter-annotator agreement.
Sense-tagged corpus: SemCor


- Subset of the English Brown Corpus – American English, collected 1963 – containing 360,000 words.

- Sense Inventory: WordNet 3.0 (also earlier ones).

- Largest publicly available sense-tagged corpus.
Statistical classification

- Supervised machine learning.

- Binary classification – positive and negative instances.

- Instances are analyzed into a set of quantifiable properties, features, defining a feature space. In WSD: aspects of context. How to “engineer” features?
  - Which properties to turn into features?
  - How to compute weights for them?
  - How to select the most useful ones?
    If we use a filtering mechanism afterwards.
The idea illustrated in two dimensions

$H_3$ is the best discriminating hyperplane.
Classifier generation and classification

- **Training**: A classifier (SVM) is generated from annotated data, i.e. feature vectors marked as positive or negative instances, by means of a training module. "Supervised" training.

- **Classification**: The classifier (SVM) is applied to unseen instances (i.e. the corresponding feature vectors) giving us a positive or negative verdict.

- Other annotated data allow us to validate a classifier. (Training data and validation data should be disjoint.)
Supervised WSD

- Data: sense-tagged corpus: i.e. collections of sense-tagged word tokens in context.

- Allows us to extract features (and thus feature vectors) characterizing tokens and contexts.

- So, we’ll extract sense-vector pairs.

- For an unseen token, we can extract the corresponding vector, and given a classifier trained on the known sense-vector pairs, predict the sense that lies closest to the vector.
Binary WSD (as in assignment proposal)

- Applies to tokens of a certain class (assignment: lemma i.e. set of forms of the same lexical item).

- Binary classification (most basic form) – Does a token belong to a certain sense or not?

- Multi-label classification (e.g. WordNet’s 8 senses of day) can be decomposed into a number of binary classification problems. (There are different methods for doing that.)

- Note that binary WSD is a simpler problem than full multi-label WSD given WordNet (for instance) if we look at evaluation scores.
Some “parameters” in doing supervised WSD

- How to produce features from tokens and their contexts?

- How to assign weights to features?

To simplify things, binary features will be assumed here, i.e. a feature is present (weight = 1) or absent (weight = 0).

There are more sophisticated weighting schemes.
Supervised WSD – engineering considerations

- How to select the best features, in case we want to reduce the number of features?

- What classifier and classifier training system to use?
  
  Assignment’s proposal: Support vector machines, viz. SVMlight (Thorsten Joachims).

- A classifier learning module typically allows different settings.
Features from tokens and their contexts

- Inflectional form of the token (if lemma instances are classified).
  E.g. *day* (singular) vs *days* (plural).

- Context features may encode e.g. form, lemma, or part of speech tag
  - defined by specific relative position (collocational features)
  - or just noted as present (in window) (bag-of-words features)

- Unigram or *n*-gram construction. (*n*-gram construction typically gives a large number of very uncommon features.)
Example (SemCor, first *day* token)

<punct>$</punct>
<wf pos=JJ lemma=10 lexsn=5:00:00:cardinal:00>10</wf>
<wf pos=IN>per</wf>
<wf pos=NN lemma=day lexsn=1:28:00::>day</wf>
<wf pos=NN lemma=increase lexsn=1:11:00::>increase</wf>
<wf pos=IN>in</wf>
<wf pos=NN lemma=expense lexsn=1:21:00::>expense</wf>
<wf pos=NN lemma=allowance lexsn=1:21:02::>allowances</wf>

(Mistagging of 10? Should be CD.)
SemCor, first *day* token

Feature identifiers (indicating their “meaning”) generated from different feature engineering principles:

- **Collocational POS features, positions -2 – +2:**
  - left2JJ, left1IN, right1NN, right2IN

- **Collocational form features, positions -2 – +2:**
  - left2_10, left1_per, right1_increase, right2_in

- **Bag-of-word-forms, positions -3 – +3:**
  - $, 10, per, increase, in, expense.
Choosing features

- We can use different kinds of features at the same time, of course.

- We can benefit from using a selection scheme for finding the most relevant ones and removing the less relevant ones. This might lead to better or (more certainly) to more efficient training and classification.

- Or perhaps we’ll work with a learning algorithm that performs well also with a large number of features (like SVM-based ones). (If we are willing to pay the computational cost for that.)
Extracting vectors for examples

- Given that we only have 1 (present) and 0 (absent) as weights, feature vectors correspond to plain sets of features.

- Features are mapped to integers before being fed to SVMlight:

  A positive (+1) and a negative example (-1):

  +1  12:1.0  17:1.0
  −1  3:1.0  5:1.0
### Evaluation metrics – Confusion matrix

Binary classification – four possibilities:

<table>
<thead>
<tr>
<th></th>
<th>positive predictions</th>
<th>negative predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>examples</strong></td>
<td>true positives</td>
<td>false negatives</td>
</tr>
<tr>
<td><strong>non-examples</strong></td>
<td>false positives</td>
<td>true negatives</td>
</tr>
</tbody>
</table>
Evaluation metrics

These are standard evaluation metrics in relation to binary classification tasks:

\[
\text{precision} = \frac{\text{truePositivePredictions}}{\text{positivePredictions}}
\]

\[
\text{recall} = \frac{\text{truePositivePredictions}}{\text{positiveExamples}}
\]

F1 score: harmonic mean of precision and recall

\[
F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]
$n$-fold crossvalidation

- Training data and evaluation data should not overlap. Evaluation data should be “unseen”.
- We need a lot of data for training, so, let’s use most of it for training and the rest for validation.
- $n$-fold crossvalidation: randomly partition the data, $D$, into $n$ (i.e. disjoint) subsets, $S_1, \ldots, S_n$, of equal size.  
  I.e. $D = S_1 \cup \ldots \cup S_n$. 
  Then, for each $1 \leq i \leq n$, perform training with $D - S_i$ and validation with $S_i$ – thus using all examples as unseen validation data in $n$ training-validation rounds.
- Common: $n = 10$ (and in assignment). “Leave one out”: $n$ is the same as the number of instances in the data, $D$ (for situations where you have a lot of time and a small collection of data).
Assignment

- Proposal: Work with a given java implementation for supervised WSD.

- We look at those senses – there are 32 – which have at least 100 positive instances, and which applies to between 40 and 60 percent of all the instances of that lemma (in SemCor).

- These are still very small collections of data for WSD.
Implementation – Data objects

se.uu.lingfil.stp.wsd.SenseDistinction
Represents a sense distinction (a classification “target”).

se.uu.lingfil.stp.wsd.SemCorToken
Holds information about a token in the corpus.

se.uu.lingfil.stp.wsd.features.FeatureSet
For a collection of features (as strings)
Implementation – Feature extraction

se.uu.lingfil.stp.wsd.features.FeatureExtractor
Interface to a specific feature extraction scheme.

Requires this method:

FeatureSet features(LinkedList<SemCorToken> sentence, int i, String cat)

For extracting features for a SemCorToken – at index i – in a list (intended to be a sentence) of such tokens.

Two simple implementations given, e.g. FeatureExtractorLetterLeft.
Implementation – Corpus data extraction

`se.uu.lingfil.stp.wsd.SemCor` contains this method for retrieving instances from the corpus:

```java
static LinkedList<FeatureSet> extractInstances(SenseDistinction senseDist,
                                              FeatureExtractor featExtr)
```

Extracts examples, as a list of `FeatureSet`s, given a sense distinction and a feature extractor.
Implementation – SvmLight

se.uu.lingfil.stp.wsd.SvmLight
– interface to the SvmLight modules. Methods:

static void learn(String datafile)

static void classify(String datafile, String predfile)

Only the last SVM is stored and used. Predictions (decision function values) written to file predfile. Decision function values – rational numbers, positive or negative reflects decision.
Implementation – Crossvalidation

```
se.uu.lingfil.stp.wsd.WsdCrossvalidation

public WsdCrossvalidation(String taskIdentifier,
                           LinkedList<FeatureSet> instanceCollection,
                           int folds)
```

Constructs an $n$-fold crossvalidation setup for the collection of feature sets (and the given SvmLight settings).
Implementation – Crossvalidation

se.uu.lingfil.stp.wsd.WsdCrossvalidation

void performAndEvaluate()
Runs the crossvalidation and counts the outcomes. (Confusion matrix: positives and negatives, true and false.) As the result depends on a random partitioning of the data into ten folds, each run will give slightly different results.

double precision()
double recall(), etc.
To retrieve the outcome scores.
Implementation – Main method iteration

for (SenseDistinction senseDist: senseDistinctions) {
    LinkedList<FeatureSet> instanceCollection = SemCor.extractInstances(senseDist, featureExtractor);

    WsdCrossvalidation svmValid =
        new WsdCrossvalidation(senseDist.expLabel(), instanceCollection, 10);

    svmValid.performAndEvaluate();
}