Word-sense disambiguation using Simplified Lesk Algorithm

Simplified Lesk algorithm

Simplified Lesk algorithm allows us to disambiguate sense of the of word, by looking at the context of the word and compare it with dictionary definition. Context can be considered terms that are within the sentence.

Pseudo-algorithm from wikipedia:

```
function SIMPLIFIED LESK(word,sentence) returns best sense of word
    best-sense <- most frequent sense for word
    max-overlap <- 0
    context <- set of words in sentence
    for each sense in senses of word do
        signature <- set of words in the gloss and examples of sense
        overlap <- COMPUTEOVERLAP (signature,context)
        if overlap > max-overlap then
            max-overlap <- overlap
            best-sense <- sense
end return (best-sense)
```

Implementation

Since algorithm does not specify how to tokenise sentence, how to normalise words, should we remove stop words or no and how to measure similarity between two set of word I added following class arguments as hyper parameter:

- similarity_measurement – function that takes two list of words and return real value
- tokenizer – function that takes string (sentence) and return list of words
- normalisation –function that takes word and return normalized word
- words_filter – function that takes list of words and return subset of these words

Extension:

I propose to add synonyms when we extract set of words from context. I used 2 approaches:

1) Add one synonym of each word in context

2) Add all synonyms of each word in context

Due to this extension there is one more hyper parameter:

- set_of_words_extractor – function that takes set of word and return super set of these words

Most common sense:

We want to compare 2 methods:

1)Simply selecting the most common sense

2)Lesk algorithm

So one more parameter:

- best_selector – function that takes list of definitions, set of words in context and similarity_measurement and returns the index of best definition

Pipeline:

1) First we preprocess context:
    a. Convert to lower case all words
    b. Filter words by words_filter(If any)
    c. Normalize each word using normalisation (If any)
    d. Extract new superset of word using set_of_words_extractor (if any)
2) For each lemma of the word
    a. We pre-process definition and examples sentences:
        i. Tokenize sentence using tokenizer
        ii. Remove punctuation
        iii. Filter words by words_filter(If any)
        iv. Normalize each word using normalisation (If any)
3) Find lemma with highest similarity with the word using best_selector (with similarity_measurement as parameter to function)

Grid Search:

I used following values for hyper parameters:

similarity_measurement:

1) jaccard_similarity – size of intersection divided by size of union
2) matched_words_similarity – size of intersection

best_selector:

1) select_best – select definition with highest similarity
2) select_first – select first definition
3) select_best_five – select from top 5 common senses

tokenizer:

1) word_tokenize - splits text on whitespace and punctuation:
2) casual_tokenize - Function for wrapping the Tweeter tokenizer.

normalisation:

1) PorterStemmer().stem - The idea of stemming is a sort of normalizing method. Many variations of words carry the same meaning, other than when tense is involved.

2) wn.morphy - Morphy uses a combination of inflectional ending rules and exception lists to handle a variety of different possibilities
3) None – don’t use normalisation at all

set_of_words_extractor:

1) WSD.synonyms_generator – Add all synonyms of each word in context
2) WSD.one_synonym_generator - Add one synonym of each word in context
3) None – Don’t produce new words

words_filter:

1) WSD.remove_stop_words –remove stop word
2) None – Don’t do any filter

Results of grid search:

Testing is done on all combination of parameters’ values and then averaged. These value is need to compare average performance of values for specific parameter.

Hyper Parameter	№1 and its average accuracy	№2 and its average accuracy	№3 and its average accuracy
set_of_words_extractor	one_synonym_generator 0.595	None 0.484	synonyms_generator 0.230
normalisation	Stem: 0.499	None 0.398	Morphy 0.41
words_filter	None: 0.495	Remove_stop_words: 0.377	
similarity_measurement	matched_words_similarity 0.494	jaccard_similarity: 0.3794	
tokenizer	casual_tokenize: 0.436	word_tokenize: 0.436	
best_selector	Select_best_five: 0.448	select_best: 0.425	

The best result 67% accuracy is obtained using following parameters:

- similarity_measurement == matched_words_similarity
- best_selector == select_best_five
- set_of_words_extractor == NONE
- words_filter == NONE
- normalisation == morphy
- tokenizer == casual_tokenize

While simply selecting the most common sense gives 66.6 % accuracy

Testing

Code for tests was extracted from:

http://www.derczynski.com/sheffield/teaching/inno/senseval.ipynb