Interface for finding close matches from Translation Memory

Intelligent Systems and Interfaces - CS565

Shrinivas Acharya (10010164)

Saurav Kumar Patel (10010159)

Sudhanshu Mittal (10010166)

Florian Kreitmair (X130101082)

# 

# Introduction

A possible option for replacing conventional rule-based machine translation by corpus-based translation that infers rules automatically, is to use a database of pre-translated sentences. For each machine translation task, a closest matching sentence from the origin language along with its translation can be obtained from the database, thus allowing to simply replace one or a few parts of the record to generate a valid translation for a previously unknown sentence. However, the complexity and thus the costs for finding good matches in a large database are high as the original sentence has to be compared with all records available in a more conventional approach.

This project aims to develop a strategy to access the closest matching sentences in the translation memory with just a small amount of computing effort needed by applying methods from Information Retrieval and Computational Intelligence. It goes on to suggest and implement in the simple metric of distance metric to incorporate the deeper sense and semantics of the words.

The approach

### The Information Retrieval Engine

Python’s whoosh library was used to construct the IR engine.

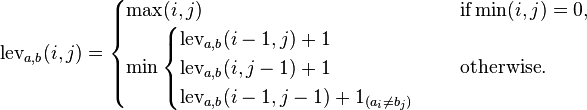
First of all we used the inbuilt indexer to build index of our translational database that will be used for querying. We take the input sentence and make a query by ORing all the words in the input sentence. Then we provide this query to our IR engine which returns a set of sentences containing some or all the query terms. This significantly reduces our search space for other parts of the program.

### Levenshtein distance

the **Levenshtein distance** is astring metric for measuring the difference between two sequences. It can be extended to find similarities between sentences by treating words as individual characters. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other.

Mathematically, the Levenshtein distance between two strings  is given by

 where



where  is the indicator function equal to 0 when  and 1 otherwise.

Note that the first element in the minimum corresponds to deletion (from  to ), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

In order to implement the method of calculating the levenshtein distance efficiently, we applied *dynamic programming* in a bottom up fashion.

Apart from being computationally expensive, the levenshtein distance also lacks in that it disregards the semantics of words while comparing them. For example-

1. *I went fishing for some sea bass. (bass refers to a type of fish)*
2. *The bass line of the song is too weak. (bass refers to tones of low frequency)*

Clearly, bass attains a different meaning in the two sentences and may have a different translation in some non-english language. But the simple levenshtein would consider both these as same, which is undesirable.

To tackle this problem we take the help of word sense disambiguation.

### Word Sense Disambiguation (WSD)

We incorporate the information gained through word sense disambiguation into the levenshtein by appending the meanings of each word to them in every sentence of the corpus. This way, we can always figure out if morphologically same words are in fact the same.

WSD refers to the process of identifying whichsense of a word (i.e.meaning) is used in asentence, when the word has multiple meanings.

The process of wsd is also applied to the words in the query sentence while calculating the levenshtein distance with some matching sentence only when they are are shared by the two sentences.

We have used lesk algorithm for the purpose of word sense disambiguation.

### Lesk Algorithm

The Lesk algorithm is a classical algorithm for [word sense disambiguation](http://en.wikipedia.org/wiki/Word_sense_disambiguation) introduced by [Michael E. Lesk](http://en.wikipedia.org/wiki/Mike_Lesk) in 1986. Various improvements for the algorithm have been proposed since then.

The Lesk algorithm is based on the assumption that words in a given "neighborhood" (section of text) will tend to share a common topic. A simplified version of the Lesk algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighborhood. Versions have been adapted to useWordnet. An implementation might look like this:

1. for every sense of the word being disambiguated one should count the amount of words that are in both neighborhood of that word and in the definition of each sense in a dictionary
2. the sense that is to be chosen is the sense which has the biggest number of this count

A frequently used example illustrating this algorithm is for the context "pine cone". The following dictionary definitions are used:

PINE   
1. kinds of evergreen tree with needle-shaped leaves  
2. waste away through sorrow or illness

CONE   
1. solid body which narrows to a point  
2. something of this shape whether solid or hollow  
3. fruit of certain evergreen trees

As can be seen, the best intersection is Pine #1 ⋂ Cone #3 = 2.

Simplified LESK Algorithm with smart default word sense (Vasileseu et al., 2004)

|  |
| --- |
| 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*) |

# Translational Database

For the purpose of evaluating our system we have used the german-english parallel corpus [European Parliament Proceedings Parallel Corpus 1996-2011](http://www.statmt.org/europarl/) , which consists of 19,20,209 sentences , 44,548,491 german words and 47,818,827 english words.

# Results

The levenshtein has proved to be quite effective in finding the closest matching sentences.

The time complexities for the various modules are-

* finding levenshtein distance between two sentences -> O(m\*n), where m and n are length of two sentences.
* getting top k results based on edit distance is O(PlogP), where P is O(S\*n^2)
  + S -> number of sentences returned from our IR engine
  + n-> maximum length of sentences

Word sense disambiguation has not been quite effective in the particular case of the english-german corpus specifically because of the fact that there is little variation in the context of the sentences described in the corpus.

Testing the effectiveness of the wsd approach is a matter of further research.

# References

[1] Li Yujian; Liu Bo, "A Normalized Levenshtein Distance Metric," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.29, no.6, pp.1091,1095, June 2007

[2] Europarl: A Parallel Corpus for Statistical Machine Translation, *Philipp Koehn*, MT Summit 2005

[3] Whoosh python documentation <http://pythonhosted.org/Whoosh/>

[4] Wikipedia lesk algorithm <http://en.wikipedia.org/wiki/Lesk_algorithm>

[5] Wikipedia Word sense disambiguation <http://en.wikipedia.org/wiki/Word-sense_disambiguation#Dictionary-_and_knowledge-based_methods>