Analyzing German Noun Compounds using a Web-Scale Dataset – Report Week 1

TECHNISCHE UNIVERSITÄT DARMSTADT

UIMA Software Project WS 2010/2011
Jens Haase

1 Work done in the last week

1.1 Dictionary

To create all possible splitting of compound word, I first need a dictionary with a list of words. Today all common spell checker, like `hunspell` or `ispell` use the `igerman98` dictionary `http://www.j3e.de/ispell/igerman98/index_en.html`. This dictionary contains over 80,000 words. Each word in this dictionary can also have optional flags. These flags can change the prefix or suffix of a word. But for our purpose we only want to work with the given list of words. More information on the file formats can be found on `http://pwet.fr/man/linux/fichiers_speciaux/hunspell`.

The dictionary file can be found at `src/main/resources/de_DE.dic`. Use the class `IGerman98Dictionary` to read and work on this file.

For corpus based dictionaries the class `SimpleDictionary` can be used. The class reads a file with a list of word. Each line must contain one word.

1.2 Access to Google Web1T

Accessing the Google Web1T corpus was not as easy as I thought. The corpus is splitted in unigrams, bigrams, trigrams, fourgrams and fivegrams. Each of these package contains one or more single archives that have to be extracted. The extracted files result in one text file, containing a sorted list of n-grams. The following listing shows the file format of one row.

```
token−1 <space> ... <space> token−n <tab> frequency
```

While the compressed size of the German corpus is 3 GB, the extracted files have a size of 11,3 GB and contain following data:

Number of tokens:	131,435,672,897
Number of sentences:	15,715,470,319
Number of unigrams:	15,133,396
Number of bigrams:	88,668,144
Number of trigrams:	154,226,178
Number of fourgrams:	140,670,210
Number of fivegrams:	100,542,274
Number of n-grams:	499,240,202

The first plan to access the data in Java was to use the jWeb1T tool (`http://hlt.fbk.eu/en/node/81`). This tool can find n-grams in a logarithmic time. But it can only find the frequency for a given n-gram. For example if the n-gram 'hello world' is listed in the corpus with the frequency 50, the jWeb1T tool will find the n-gram by searching for 'hello world', but not by searching for 'hello' or 'world'.

For the weighting function I want to search for 'hello' and get all n-grams containing the word 'hello'. When I search 'hello world' I want to find the all n-grams with the words 'hello' and 'world'.

A second tool is the Java API for Web-1T Corpus of Digital Pebble `http://www.digitalpebble.com/resources.html`. The tool is currently not free available, but it is possible to contact Digital Pebble. According to Julien Nioche of Digital Pebble the tool is comparable to jWeb1T, but they are currently working on a new version that will be freely available soon. Since I have not the time to wait for this tool I have to create my own indexer.

I simply used lucene to index all n-grams. One n-gram is one document in the lucene index containing the n-gram and the frequency. All data is stored in the index so the extracted Google Web1T files are no longer needed. Creating an index can be done with following command in the projects folder

```
./bin/web1TLuceneIndexer.sh \
  --web1t PATH/TO/FOLDER/WITH/ALL/EXTRACTED/N-GRAM/FILES \
  --outputPath PATH/TO/LUCENE/INDEX/FOLDER
```

But keep in mind that the task takes very long a writes a huge amount of data to the hard disk. For me the task runs around 5 hours (313 minutes). The index has a size of 23,4 GB.

Using this tool for the English corpus will fail because Lucene can only handle 2,147,483,647 (Integer.MAX_VALUE) documents. In the English corpus are 3,795,790,711 n-grams.

To access the generated index the class `Finder` can be used.

2 Plan for next week

In the next week I will focus my work on the splitting algorithm. The splitting algorithm should split a word in all possible decompound forms. To test the algorithm I will try to create a large list of words, with all possible splittings.

For the split algorithm I also need a list of linking morphemes. These can either be hard coded in Java code or in a separated text file.