MLP, Ling 331

Solutions to Exercise 1

1 Tokenization

Goal: Tokenization is used to identify all tokens, i.e., "word-like chunks" (Beesley, Karttunen.2003:422) such as words, multi words expressions (MWE), and punctuations in a text. I

How does it work? In languages like English and German, words can be understood as strings in between two empty spaces ("white-space strategy"). Regarding multi-words expressions, examples like *New York, New Yorker,* as well as *Mr. Miller* or *Krueger-Johansson* need to be captured. With respect to punctuations, it is not only commas, semi-colons, periods and alike, but also abbreviations like *e.g.*, *i.e.*, or *U.S.A.*

The white-space strategy does not work for languages that do not have special characters for word boundaries (e.g. Chinese, Japanese or Classical Latin). These needalternative ways to identify word boundaries. Chinese often uses a mixture of different strategies, including, but not limited to, statistics, n-gram segmentation and lexicons. Generally, the maximum match strategy is used. This strategy is illustrated on the basis of the English example *thetabledownthere* in J&M.

Concretely, tokenization is a problem that has the complexity of a finite-state machine and can thus be dealt with well via a FSA. An example is the approach by Beesley and Karttunen we discussed in class.

The tokenizer takes strings or entire files of text as an input and transduces it to get one token per line as an output. First of all, the program needs definitions of white spaces, which can be an empty space, tab or a new line. Also, special characters (e.g. ", §, $,], }, etc.) and punctuations (i.e., ., ,, `,`,') need to be defined, as well as alphabetical characters. With the help from regular expressions, words can be defined easily, namely by stating that a word consists of at least one character, whereby a character can be both alphabetical and punctuation symbols. Same holds for abbreviations, which at least consist of one letter and one dot. After having defined both upper and lower case letters, a regular expression would look like [letter \.]+, where *letter* is a variable for the letter-definition and "\" protects the following dot. Since the expression is put in brackets and the Kleene + follows this expression, and abbreviation consists of at least one letter followed by a dot. Infinite repetitions of this sequence are allowed to follow.

Definite abbreviations such as *Mrs., Ms., Mr., Dr., Ltd.*, etc. need to be defined separately. Various numbers such as integers and floating numbers both positive and negative, again, can be expressed by regular expressions. Hence, such an expression would show that a number consist of at least one digit, plus/minus sign, or a separating sign and at least one following digit. Thus, possible results could be: +2; -2; +.23; -,4; 6.987;-5.0; +2,87, etc. Notice that with this regular expression multi-figure integers are illegal.

Multi-word expressions are a bit more demanding; covering MWE by simply stating

them in a variable, would be very complex since inflected forms such as *New Yorker* from *New York* would need an extra listing, too, in order not to split up the actual MWE into *New Yorker + er*. Hence, one regular expression which should cover such cases would recognize MWE and and would allow additional characters up to a white space.

Tokenized texts consider all those restrictions and rules and generally output one token per line.

Concrete Output: The Xerox tokenization tool works well and as expected. The possessive 's is treated as an individual token. This is standard in NLP approaches.

2 POS Tagging

Goal: Provide information about word class/token type for individual tokens in a text.

How does it work? 1) A tag set needs to be defined. These tend to be language specific and require and understanding of the linguistic structure. Famous tagsets are the Penn Treebank tagset, the CLAWS tagset and the German STTS tagset.

One can tag a corpus manually (expensive and prone to human error). One can use automatic methods which can be rule-based or using stochastic methods or a combination of both. The Brill tagger is a famous tagger that used to be primarily rule-based. Most current taggers are based on machine learning algorithms, factoring in information about syntactic distribution via a calculation of ngrams.

Concrete Output: The outputs are fairly good for both English and German. However, the taggers do make mistakes.

<u>English:</u>

```
age
    age +NOUN
 of
    of +PREP
 one
    one +PRONONE
```

"one" is not a pronoun, but a number.

```
the
    the +DET
 instant
    instant +ADJ
```

"instant" is a noun

whose
who +DETREL
name
name +VPRES

"name" is a noun

German:

einem
ein +ART
Jahr
Jahr +NOUN

"einem" is a number.

überlebte
überleben +ADJA
Harry
Harry +NOUN

"überlebte" is a verb

noch
noch +ADV
Angst
angst +ADV

"Angst" is a noun.

Generally, only the first part of a compound is returned. A good treatment of compounding remains a problem for even the best POS taggers.

3 Morphological Analysis

Why useful? In deep NLP one would like to have more information about a word as provided by the morphology. In QA systems, morphological information may provide crucial information for inferencing (if something happened to many cats (plural) then one can infer that it happened to at least one cat (singular)). For shallow NLP one would like to be able to abstract away from the different morphological realizations of a word and do information retrieval similarly, for example, for *dog* vs. *dogs*.

Porter Stemmer: Porter Stemmers main advantage is their ease of usage. They are both easy to understand and to implement. They are also computationally efficient, given the simplicity of their code. They are not linguistically "correct", but they efficiently and quickly provide some kind of a stem for a family words. This "stem"

can then be used for information retrieval purposes.

Disadvantage is that information coming from the morphology is thrown away and that the simplicity of the code makes them error prone. One has cases of overstemming (e.g. *wander* → *wand*), or missteming (e.g. *relativity* → *relative*) or understemming (e.g., *greatest* → *greatest*) and it also does not recognize proper names, as *Harry* in the sample text.

Xerox tools: The morphological analysis by *Xerox* uses a finite-state two-level morphological analysis. The output is mainly as expected. It recognized proper names, although it could not identify the masculine gender of *Voldemort*. This, however, was expected, too, since the morphological analysis does not take into account the analyses of the current neighbors. Unexpectedly, the analyzer defined the possessive *'s* by *'s +open+NOUN.* Past tense verbs are ambiguously adjectives in some cases to reflect instances like *The car was completely destroyed.*

3 Parsing

Grammar Fragment

German: Only the a sentence can be parsed.

Lexical entries need to be added for: *bellen*

English: None of the sentences can be parsed.

Lexical entries need to be added for: *sees, bark.*

Grammar Update:

S → NP VP
NP → (D) ADJ* N PP*
VP → V (NP) PP*
P → P NP

Kleene Expressions

Kleene *: strings of 0 to infinite elements that precede the Kleene *.
Kleene +: strings of 1 to infinite elements which precede the Kleene +.

Parsing Strategies

This was done more or less right by everybody (no absolutely "right" solution).