Interactive Technologies – Exercise 3

Christopher Köllner, Michael Plattner, Matthias Ringwald

Description

Goal of this exercise was to take a database of text messages collected by the University of Singapore and use it to learn a prediction algorithm that suggests possible succeeding word using the last word and suggests word completions using the already typed in prefix. This should be tested in a simple GUI.

Training the prediction models

We split the algorithm into two separate prediction models, one is used for prediction the succeeding word, one is used for word completion.

Succeeding words

The training it self works similar for both, but let's explain this with the example of training the succeeding word predictor. For all the text messages in the database, we split up the messages into the single words by using following logic:

```
String[] words = text.replaceAll("[^a-zA-ZäöüÄÖÜ ]", "")
                     .toLowerCase().split("\\s+");
```

For training the succeeding word predictor we then always take two succeeding words and put them into a big encapsulated HashMap. The outer HashMap uses the first word as key and holds the inner HashMap as value. The inner HashMap uses the second word as key and stores an Integer value. This Integer value represents the count of how often in the database the first word is followed by the second word. By running through all the text messages, the right counter is always increased by 1. This can be done efficiently, because accessing the right counter just uses two accesses to HashMaps.

```
HashMap<String, HashMap<String, Integer>>
```

Using this data structure for calculating the most probable succeeding words, however, is not that efficient, because we would need to run through all entries of the inner HashMap to find the best suggestions. Therefor we convert the trained data structure into another one that is more efficient for finding the best suggestions. We keep the outer HashMap with the first word as key, but as value we use a List of Map.Entry objects. This way we can take all the inner HashMaps, extract their entries, sort them using the stored number of occurrences and save them as list. This sorting has to be done just once for all the inner HashMaps.

```
HashMap<String, List<Map.Entry<String, Integer>>>
```

When we then need suggestions for a succeeding word, we just access the HashMap with the last written word as key, which is very efficient, and we get a sorted list of possible succeeding words. Now we just have to take the most probable ones and show them in the GUI.

Word completion

The data structures and training method for word completion is very similar to the succeeding words method. The data structures are exactly the same, the only difference is, that the keys of the HashMaps in this case are not the first word and the second word, but the prefix and the complete word. We take all the words that appear in the database and feed them into our data structure multiple times using different prefix lengths. First, we just use the first letter of the word a s prefix,

next we use the first two letters as prefix and so on, until the prefix equals the whole word. We initially also implemented a limit, for example to only feed a maximum of 3 letters per prefix into the database, but we found out that the training process won't take much longer if we just use all possible prefixes and the user experience is much better if the GUI adapts to every single keystroke and doesn't stop after the first 3 letters of a word.

GUI

The GUI looks like follows. We have one big TextArea where thy user can type his text, on top of it we have a button panel which shows all the suggestions of the algorithm. When pressing on of the suggestions with the mouse, the word is added to the written text. With every change of the TextArea the suggestions are updated.

If the last symbol of the text is a space, the most probable succeeding words are shown, otherwise, word completions are shown. Initially, when the text area is empty, the GUI just suggests the word "Hello".