Decoding T9 (predictive text)

You may remember that entering text on cell phones used to work with only 9 buttons. Early systems made you press the same button multiple times to select the exact character you want. The T-9\textsuperscript{TM} system (“Text on 9 keys”) uses only one keypress per character. Then the system has to use this limited information to somehow figure out what the user meant: for example that 443336 might mean HIDDEN instead of GGDDDM or HIEDFO. These systems are based on a word list\footnote{https://www.youtube.com/watch?v=6hcoT6yxFoU}, and the word list has to be quite large: a megabyte (uncompressed) is not unreasonable. (Just a normal dictionary is not enough, because of conjugations and declensions.) In this exercise you will prototype a system that uses only a few kilobytes of storage.
Markov models for words

Consider words in some language: strings of characters. We can build a Markov chain to model these words: the states are the possible characters, and the transitions are “which character comes next.” For example, in English we might expect that a T is much more likely to be followed by an H than by a Z. That is, in English we expect:

\[ \Pr[X_{n+1} = H \mid X_n = T] > \Pr[X_{n+1} = Z \mid X_n = T] \]

In German the transition TZ might be more likely, so the actual probabilities in the Markov model will have to correspond to the language we are trying to model.

(a) How can we get reasonable values for these probabilities (initial and transition)? How many probabilities do we need to know? How does this reduce the amount of information that we need to store? (1 Punkt)

A Hidden Markov model for T9

Now we will use a hidden Markov model to find a most-likely estimate for the word the user tried to type. As input we get a string \( S_{in} \in \Sigma^*_{\text{keys}} \), where \( \Sigma_{\text{keys}} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \).

As output, we want to find a most-likely explanation of these keypresses: a string \( S_{out} \in \Sigma^*_{\text{real}} \), where \( \Sigma_{\text{real}} = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, \_\} \).

“Most likely” here is according to the hidden Markov model.

(b) What observation probabilities should we use? How do we get them? (1 Punkt)

Basic decoding

Write a program that can do the following. Use a reasonable programming language; do not use special libraries.

- Convert a word to a sequence of keypresses. That is, convert from \( \Sigma^*_{\text{real}} \) to \( \Sigma^*_{\text{keys}} \).
- Given a sequence of keypresses, calculate the most-likely intended word. That is, convert from \( \Sigma^*_{\text{keys}} \) to \( \Sigma^*_{\text{real}} \).
- Learn the parameters of the model from an example text.

(c) Hand in the code and any data for your implementation. Make sure the source code is reasonably readable. Describe the hidden Markov model you use in the comments. (5 Punkte)

---

2The lecture slides talk about topological orders, for example, but that is the general case for LONGEST PATH on directed acyclic graphs; your program does not need to calculate one, since for this specific problem you already know a topological order. It is not necessary to explicitly create the network described in the slides. Certainly don’t use a library for maximum-likelihood estimation in hidden Markov models.
Improvements

A problem with the basic system is that it has a tendency to generate nonsense words. (After all, it doesn’t have a dictionary.) If your implementation of basic decoding works, try to improve the system. Perhaps you can change the alphabet (include Ü, ß, . . . ?); perhaps design a better Markov chain, possibly with different (more?) states or transitions (but how does that influence storage space?); perhaps redistribute the mapping of characters to keys; &c.

(d) Describe and/or implement such improvements. Does it actually work better? (“No” is a valid conclusion.) Make sure to include a version of the program that does just the basic decoding. (7 Punkte)