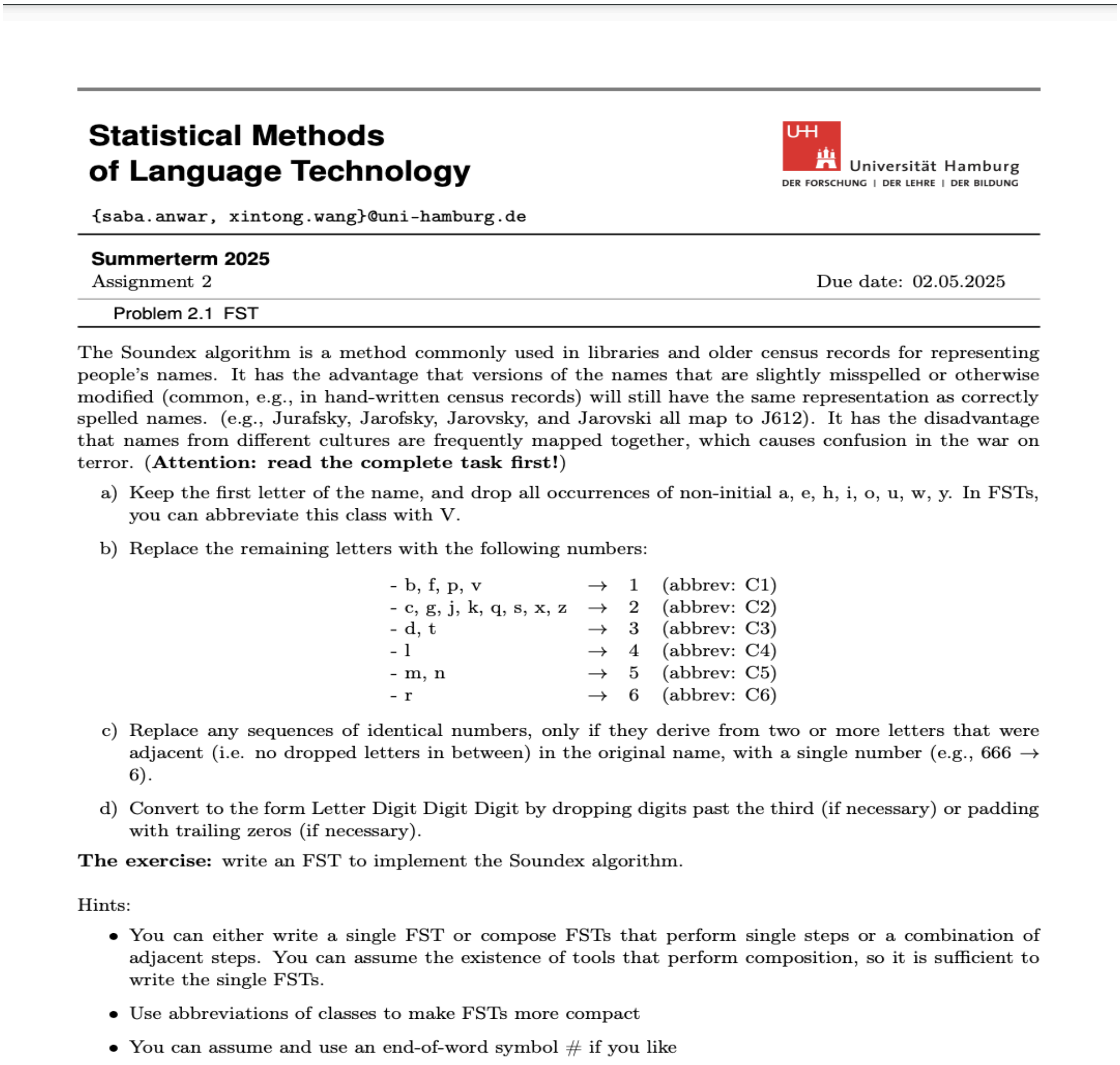


Statistical Methods of Language Technology

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

{saba.anwar, xintong.wang}@uni-hamburg.de

Summerterm 2025
Assignment 2 Due date: 02.05.2025

Problem 2.1 FST

The Soundex algorithm is a method commonly used in libraries and older census records for representing people's names. It has the advantage that versions of the names that are slightly misspelled or otherwise modified (common, e.g., in hand-written census records) will still have the same representation as correctly spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map to J612). It has the disadvantage that names from different cultures are frequently mapped together, which causes confusion in the war on terror. (**Attention: read the complete task first!**)

a) Keep the first letter of the name, and drop all occurrences of non-initial a, e, h, i, o, u, w, y. In FSTs, you can abbreviate this class with V.

b) Replace the remaining letters with the following numbers:

- b, f, p, v	→	1	(abbrev: C1)
- c, g, j, k, q, s, x, z	→	2	(abbrev: C2)
- d, t	→	3	(abbrev: C3)
- l	→	4	(abbrev: C4)
- m, n	→	5	(abbrev: C5)
- r	→	6	(abbrev: C6)

c) Replace any sequences of identical numbers, only if they derive from two or more letters that were adjacent (i.e. no dropped letters in between) in the original name, with a single number (e.g., 666 → 6).

d) Convert to the form Letter Digit Digit Digit by dropping digits past the third (if necessary) or padding with trailing zeros (if necessary).

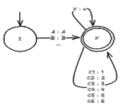
The exercise: write an FST to implement the Soundex algorithm.

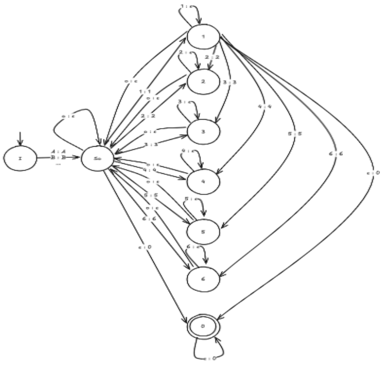
Hints:

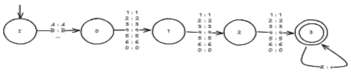
- You can either write a single FST or compose FSTs that perform single steps or a combination of adjacent steps. You can assume the existence of tools that perform composition, so it is sufficient to write the single FSTs.
- Use abbreviations of classes to make FSTs more compact
- You can assume and use an end-of-word symbol # if you like

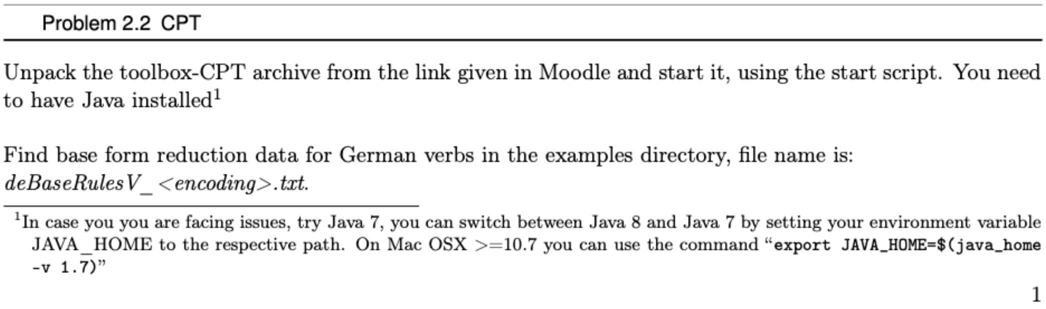
Code: Github

The composition of these three FST gives a FST that implement the Soundex algorithm. Note that in the second FST, the numbered states 2, 3, 4, 5 and 6 have similar transitions to the other numbered states as state 1. For readability purposes, these transitions have not been represented here.







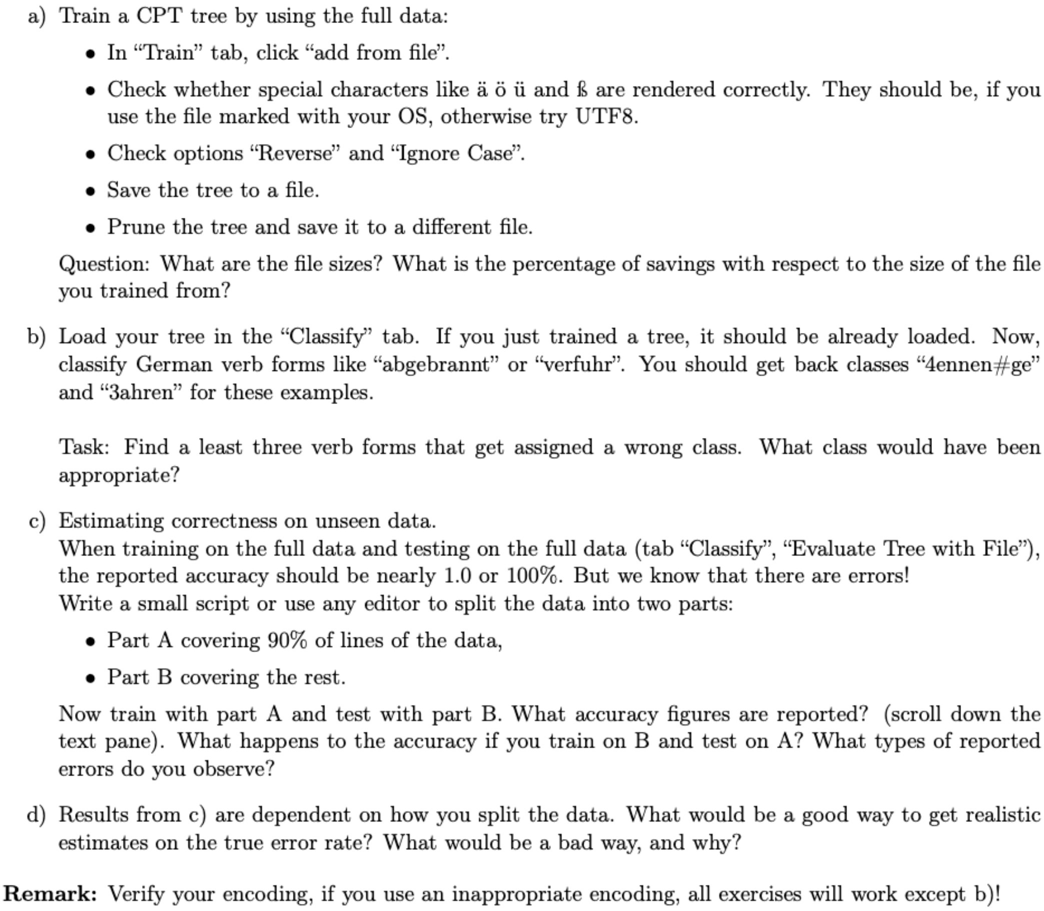


Problem 2.2 CPT

Unpack the toolbox-CPT archive from the link given in Moodle and start it, using the start script. You need to have Java installed[1]

Find base form reduction data for German verbs in the examples directory, file name is: *deBaseRulesV_<encoding>.txt*.

[1] In case you you are facing issues, try Java 7, you can switch between Java 8 and Java 7 by setting your environment variable JAVA_HOME to the respective path. On Mac OSX >=10.7 you can use the command "`export JAVA_HOME=$(java_home -v 1.7)`"



a) Train a CPT tree by using the full data:
   - In "Train" tab, click "add from file".
   - Check whether special characters like ä ö ü and ß are rendered correctly. They should be, if you use the file marked with your OS, otherwise try UTF8.
   - Check options "Reverse" and "Ignore Case".
   - Save the tree to a file.
   - Prune the tree and save it to a different file.

   Question: What are the file sizes? What is the percentage of savings with respect to the size of the file you trained from?

b) Load your tree in the "Classify" tab. If you just trained a tree, it should be already loaded. Now, classify German verb forms like "abgebrannt" or "verfuhr". You should get back classes "4ennen#ge" and "3ahren" for these examples.

   Task: Find a least three verb forms that get assigned a wrong class. What class would have been appropriate?

c) Estimating correctness on unseen data.
   When training on the full data and testing on the full data (tab "Classify", "Evaluate Tree with File"), the reported accuracy should be nearly 1.0 or 100%. But we know that there are errors!
   Write a small script or use any editor to split the data into two parts:
   - Part A covering 90% of lines of the data,
   - Part B covering the rest.

   Now train with part A and test with part B. What accuracy figures are reported? (scroll down the text pane). What happens to the accuracy if you train on B and test on A? What types of reported errors do you observe?

d) Results from c) are dependent on how you split the data. What would be a good way to get realistic estimates on the true error rate? What would be a bad way, and why?

Remark: Verify your encoding, if you use an inappropriate encoding, all exercises will work except b)!

2.2

a

The trained tree is 6MB before pruning, 832.5kB after. When compared to the 2.1MB file that was used as input, that makes for a memory saving of approximately 61%.

b

The verb "umspannen" should be "0" but is "0;5innen". v"erführe" should be "0n" instead of "0n;4ahren" and "schalte" should be "0n" instead of "2en;0n"

c

Training with dataset A and testing ob B gives an accuracy of 93.1%. The inverse brings the accuracy down to 76.3%.

We observe that the errors are mostly on unseen verbs that have specific exception forms, hence why the tree trained with B has seen these special forms less often.

d

By taking a wide spread of different verbs and tense we get better results. A good way of doing

so with an uninformed algorithm might be to simply sample the lines kept for the training dataset randomly. Bad sampling strategy would include taking in the alphabetical order (most tense seen but fewer diversity in the base forms) or taking a panel of different verbs but all in the same tense (as an example taking the first 4 lines for each verb).