Assignment 3
Chatbots and task-oriented agents

Problem 3.1 Information-retrieval chatbot (15p, mandatory)

In this problem you will build an information-retrieval (IR) chatbot in C#, starting from the base code available in the `StartingPoint3.zip` file that can be found under the page for Assignment 3. In this case, the starting point is just a rather simple skeleton code; you need to write quite a bit of C# code to complete the program.

The chatbot should make use of TF-IDF embeddings, as described in Section 5.3 of the compendium. You must generate your own dialogue (pair) corpus using transcripts of spoken dialogue, from the Cornell Movie Dialogue Corpus (see the link in the file `Assignment3Links.txt`). The processed data should consist of a (large) number (N) of sentence pairs (S_1, S_2), such that sentence S_2 was given in response to sentence S_1. That is, the two sentences should (i) be consecutive in a dialogue; (ii) concern two people who are actually talking to *each other*; and (iii) should not be too long, at most (say) 15-20 words each, often shorter. Thus, for this problem, it is important to extract data with care, in order to make sure that the sentence pairs fulfil the three conditions just mentioned. For the Cornell data set, there is a file that will help you fulfil these conditions; see also the comment in the code (the `DialogueCorpus` class).

Once your program has loaded and tokenized the corpus, it should compute normalized TF-IDF vectors for all sentences S_1 (i.e. for the first sentence in every pair). The length of those vectors is given by the number of distinct tokens (words) in the corpus. You must thus write code for text preprocessing, tokenization, and TF-IDF computation. You may use your own code from Assignment 1 to the extent that it is possible.

Next, when running the chatbot, for a given input sentence T, compute the cosine similarity between T and the first sentence (S_1) of *every* sentence pair (S_1, S_2) in the corpus, and sort the result (along with the index of the sentence pair in question) in decreasing order of cosine similarity. Then, rather than just selecting the pair with highest degree of similarity as in Eq. (5.6) in the compendium, select a sentence pair randomly from the top five pairs, so that the chatbot may display a bit more variability. From the selected sentence pair, return the sentence S_2 as the output from the chatbot (shown on-screen).

Note: When computing the cosine similarities, just ignore any (rare)

words in the test sentence T that do not appear in the corpus. Do *not* modify the corpus once the chatbot has been loaded.

What to hand in You should hand in the full C# code for your chatbot, as well as the (processed) text file (which you should save from your C# program) containing the corpus, i.e. all sentence pairs (one pair per row, with the two sentences separated by a tab character). Moreover, you should write a section (3.1) in your report where you briefly describe your implementation and also provide a brief manual for how to run your program, i.e. the steps required to (1) load the raw data file(s), (2) process the raw data (preprocessing, tokenization, ...) to form the set of sentence pairs (making sure to include only sentences that form actual *pairs*), (3) computing the TF-IDF vectors, and (4) running the chatbot. *Failure to provide a clear description of this kind, or omitting the data files, will result in the problem being returned, with a point loss as described below.* Finally, in the report, include also a set of sample dialogues between the chatbot and a user (you).

Evaluation For this problem, the implementation is worth 11p and the report 4p. In particular, we will assess how well your corpus fulfils the conditions (i)-(iii) above. If you need to resubmit the problem, a maximum of 9p will be given.

Problem 3.2 Black boxes and common sense reasoning (6p, mandatory)

In only the last few months several new transformer-based black-box chatbots have been released, e.g. Meta Galactica, chatGPT and, most recently, Bard (not yet publicly available). Having been trained on very large data sets, these impressive chatbots are capable of general discussions on almost any topic. They are generative in nature, formulating their output token-by-token using stochastic correlation, taking into account large chunks of contextual information. However, they lack what one may call a world model, which (if present) could provide common-sense understanding. For this reason, they can also fail spectacularly.

In this assignment, you should first generate a *successful* interaction with chatGPT, on any topic you like. You should ask the chatbot a question and, after receiving a correct answer, ask it to explain how it arrived at the answer, and again receive a correct answer. You may have to try a few examples to make this happen.

Next, you will try to find (at least two, at most five) *examples* of chatGPT failures, i.e. cases where it fails to answer a question for which the answer would be obvious to any adult human using common-sense reasoning. In the `Assignment3Links.txt` file, you will find some links to pages listing various examples of the kind just described. Your task, then, is to generate *similar* examples (but *not* identical ones - it is not allowed just to copy-paste from the web pages in question). Instead, try to formulate a common-sense reasoning task that is a simple as possible, yet makes chatGPT give an incorrect answer. You may wish to study the concept of Winograd schemas (see the links in `Assignment3Links.txt`.

What to hand in

Add a section (3.2) in your report, where you clearly specify both the entire successful dialogue and the set of dialogues where chatGPT failed to give the correct answer. For those examples, describe also the correct answer, and how you would arrive at it, using common-sense reasoning.

Evaluation The text in the report is worth 6p. If you need to resubmit the problem, a maximum of 3p will be given.

Problem 3.3 Generic cognitive processing (6p, voluntary)

Generic cognitive processing, in the form of sequences of cognitive actions, is a central concept in DAISY. A sequence of cognitive actions can be specified in the form of pseudo-code, as described in Chapter 7 in the compendium and in the slides from the corresponding lectures.

Consider now the example of a car dealing company. We will assume that a knowledge base is available, containing one data item for each specific vehicle, in the format used in DAISY (see also Fig. 7.3 in the compendium). Here, the tag-value units involve the brand (e.g. BMW, Volvo, and so on), the model type (e.g. 325), the year (when the car in question was manufactured, e.g. 2020), the price in SEK (e.g. 300000), and so on. Now, using the cognitive actions described in the slides from 20230215, specify the cognitive processing (using pseudo-code; see Section 7.2.3), required for answering the question *Do you have any $brand from $year?* (e.g. *Do you have any BMW from 2020?*) keeping in mind that the number of such vehicles *might* be zero. You do not have to specify the output, but you need to specify the target(s) of the final cognitive action, depending on the number of cars available (the preceding actions are assumed to be executed in sequence).

Next, consider a case where the user instead asks *Do you have any $brand from $year or later, priced at or below $priceLimit?*, e.g. *Do you have any BMW from 2015 or later, priced at or below 200000 SEK?* Again using the cognitive actions described in the slides from 20230215, specify pseudo-code for answering the question. Hint: The `Extract` action can handle inequalities.

What to hand in

Add a section (3.3) in your report, where you provide a full account of your solution to both exercises above, with clear pseudo-code and an associated description of the steps, for both examples.

Evaluation The text in the report is worth 6p. Resubmissions are not allowed.

Problem 3.4 DNN-based chatbot (8p, voluntary)

In this problem you will run through a Python script to generate a seq2seq RNN-based chatbot (with GRU units). The script can be found at `https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/chatbot_tutorial.ipynb`

Here, you will again use the Cornell Movie Dialogue corpus, but you need to download it from a different source (than for Problem 3.1 above), due to some detailed data format requirements in the Colab script. See the file `Assignment3Links.txt` for a full description.

Once you have downloaded and placed the data sets in the correct folder (under Colab), run through the entire Python notebook to generate the chatbot (note that you need to uncomment one line towards the end of the notebook to activate the chatbot).

Then, run through several conversations with the chatbot, and *compare* its performance (subjectively) to the performance of the information-retrieval chatbot from Problem 3.1 above.

What to hand in You should save the entire notebook (from Colab) (File - Download - Download .ipynb), showing that you have indeed run through all the steps. Moreover, in the report, add a section (3.4) with the detailed comparison of the dialogue, i.e. comparing the chatbot from this assignment with the chatbot from Problem 3.1. Moreover, based on the description in the Python notebook, make a figure showing the structure of the encoder-decoder network that defines the chatbot, with as much details as possible.

Evaluation For this problem the Python notebook is worth 3p and the report 5p. Resubmissions are not allowed.