Assignment I: Automated Correction of Incorrectly Recognised Text

Introduction

The goal of this assignment is to model and solve a practical problem using search-based techniques.

Consider a scenario where a company is digitising its hand-written paper records containing text in English. The person carrying out the task scans the paper document and then uses an OCR (Optical Character Recognition) system to extract the text. The output of the OCR system is highly noisy and hence, the characters may be incorrectly extracted. For example, if the the original text in the document was *"i am studying ai"* then it may be extracted as *"i an studying aj"* with errors occurring for two characters. Consequently, the digitised document contain sentences with imperfections and may not be proper sentences in English.

In this assignment, we apply search-based methods for determining a possible correction for the input erroneous text.

Problem Definition

The problem takes as input a sentence (sequence of all lowercase words separated by white spaces). The words in the sentence are all lowercase and may be have been erroneously extracted during the digitisation process. The output consists of a sentence which represents a "good" correction for the problem.

```
Input erroneous sentence: "i an stydling artificial intelligence" Output corrected
sentence: "i am studying artificial intelligence"
```

Further, it has been observed from prior experience that the OCR commonly confuses certain characters with certain other characters that resemble in shape, orientation etc. For example, the character "a" is incorrectly extracted as "e", "i" or "o" and less often as "l". This information regarding the possibilities for incorrect character replacement for a character is given in a file called ***conf_matrix.json.*** The format is shown below.

```
a → [e, i, o] p → [m, b] ... ... s → [c, t, z, x]
```

Note that given an erroneous string and the information regarding possible character errors, there may be a large number of possible corrections for a given input. A search based algorithm will need to explore this space of possible corrections. The algorithm will base its decisions based on a cost *function* inversely related to how likely (or fluent) the sentence is in the English language. The intuition is that lower the score, the higher is the likelihood for it to be error-free. For example, the sentence ""*i am studying ai"* is a more fluent (or more likely) English sentence in comparison with *"i an studying aj",* therefore, the former has less cost and hence is more likely to be error free.

In this assignment, the cost function is provided to you. Internally, the cost function is realised by a language model which essentially determines the fluency of the sentence as discussed above. The detailed understanding of language models is beyond the scope of this assignment.

Formally, let $f_{cost}(.)$ denote the cost function that takes an input sentence s consisting of a sequence of words $\{w_0, w_1, \ldots w_{n-1}\}$ and outputs a cost. In this assignment, the cost is evaluated as $f_{cost}(s) = -log\big(p(w_0, w_1, w_2) * p(w_1, w_2, w_3) * ... * p(w_{n-3}, w_{n-2}, w_{n-1})\big)$. Here, the cost function determines the cost as an accumulation of probabilities of *three* consecutive words at a time. This cost evaluation is provided in the file **language_model.py** as part of the starter code and can be called directly by your algorithm.

Finally, we can assume that there is a maximum time available for your algorithm to run. This is specified as the parameter ***timeout***.

Your Implementation

- Please implement a search-based algorithm for the problem described above.
- Please identify a representation for the state, operator, transition costs and the notion of a goal or a solution for this problem. Analyse the branching factor for this problem.
- One approach is to try methods such as DFS, BFS or d*ynamic programming*. These approaches are unlikely to scale beyond sentences with ~40 characters (including spaces). Vanilla A* may be difficult to apply as there is no single goal state and may still not scale.
- Alternatively, local search methods can scale to larger problems such as correcting a sentence with ~100 characters in about 10 seconds. Implementing local search would require thinking about how to consider and evaluate neighbours. The downside of local search is the difficulty of adequate exploration of the state space.
- Your own algorithm can borrow ideas from both families of approaches.
- Please rely only on search-based methods (covered in class) or their extension/adaptation. Please do not directly solve the problem directly linear/integer programming or a domain-specific technique from natural language processing beyond search. In case of a doubt in this regard, you may contact the TA to clarify.
- Your assignment submission will consist of a single submission that of your best algorithm for this problem.

Implementation Guidelines

- The starter code is provided for your implementation. Please download the code from Moodle. The zip has following structure.

```
A1 ├── data │   ├── conf_matrix.json # Confusion matrix of character replacements │
├── corpus.txt # Additional strings without errors which can be used for experimentation
│   ├── input.txt # Sample input file consisting of erroneous strings │   ├──
lm_model.pkl # Pre-trained language model (in .pkl format) that realises the cost
function. │   └── output.txt # Expected output strings for above input strings ├──
environment.yml # Conda environment file ├── language_model.py # Code to compute cost
function for the assignment. ├── run.py # A driver code for running your implementation.
├── solvers.py # The file where you place your implementation. └── utils.py # Code for
timeout utilities
```

- You can run the driver program using following command. It will run your solution for each string in the data/input.txt file and write the solutions to data/pred.txt. For each input sentence, your program is allowed to run for at most 2 seconds (as specified by the -tm option). You may experiment with this parameter.

```
python run.py -src data/input.txt -tar data/pred.txt -tm 2
```

- The file **solver.py** provides a template for your implementation. You are expected to implement the **search()** method in the class **SentenceCorrector**.
- **You ONLY need to modify the file solver.py with your implementation. Please do not modify other files.**
  solver.py is structured as below. search() method is entry point for your implementation. Note that once the timeout occurs, your execution will terminate for the current string. It's your responsibility to store the best string you have got so far into **self.best_state** variable. For more details you can check how driver program reads the output of your implementation.

```
class SentenceCorrector(object): def __init__(self, cost_fn, conf_matrix):
self.conf_matrix = conf_matrix self.cost_fn = cost_fn # You should keep updating
following variable with best string so far. self.best_state = None def search(self,
start_state): """ :param start_state: str Input string with spelling errors """ # You
should keep updating self.best_state with best string so far. # self.best_state =
start_state raise Exception("Not Implemented.")
```

- For any string, **self.cost_fn** method provides a cost based on the language model. You invoke this function as:

```
cost = self.cost_fn(state)
```

- Additionally, **self.conf_matrix** contains the list of incorrectly recognised characters as discussed before. You can access incorrectly recognised characters for input character ch as:

```
incorr_chars = self.conf_matrix[ch]
```

- Please use **Python 3.7** for your implementation. Your code should use only standard python libraries. Please do not include any dependencies on third party libraries or existing implementations of algorithms. Starter code has been updated to user Python 3.7. Do not worry if you are using Python 3.6. Your code should be compatible with Python 3.7 as well.
- A **conda** environment is provided with the allowed dependencies for this assignment so that mingling with system python or any other python libraries that you may already have involved can be avoided. The instructions for the first time conda installation is here. In general following command should create the required conda environment using **environment.yml** file provide with the starter code.

```
conda env create -f environment.yml
```

- The use of global variables or threading libraries is not permitted in your implementation.

Evaluation

- The evaluation may be on a different data set, cost function and word replacement information with different time limits.
- The output solution will be evaluated based on the quality (cost) of the output correction sentence.
- The evaluation will consist of a an *absolute* component (in relation to a baseline implementation) and a relative component (that takes into account other submissions).

Submission Instructions

- **This assignment is to be done individually or in pairs. The choice is entirely yours.**
- **The assignment is intended for both COL333 and COL671 students.** If the assignment is being done in pairs then the partner is to be selected only within your class. That is COL333 students should be paired within COL333 and similarly masters students in COL671 should pair within their class of COL671.
- Please submit only single algorithm (your best algorithm). Please describe the core ideas of your algorithm in the text file called **report.txt** (max. 2 pages in standard 11 point Arial font). The first line of the report should list the name and entry number of student(s) who submit the assignment.
- Please create your **submission package** as follows:
  - Submit a single zip file named *<A1-EntryNumber1-EntryNumber2>.zip or <A1-EntryNumber>.zip*. Please use the *"2019CS———"* style format while writing your your entry number.
  - Upon unzipping, this should yield a single folder with the same name as the zip file
  - Inside the folder should be a python file named - **solvers.py** and a **report.txt**.
  - You need not submit run.py, language model.py.
- The assignment is to be submitted on **Moodle**. Exactly ONE of the team members needs to make the submission.
- **The submission deadline is 5pm on Thursday 22.09.2022.** Late submission with penalty is allowed till 5pm on Saturday 24.09.2022. Check guidelines below for details.
- **This assignment will carry 12% of the grade.**

Other Guidelines

- Late submission deduction of (10% per day) will be awarded. Late submissions will be accepted till 2 days after the submission deadline. There are no buffer days. Hence, please submit by the submission date.
- Please strictly **adhere** to the input/output syntax specified in the assignment as the submissions are processed using scripts. Please do not modify files beyond the files you are supposed to modify. Failure to do so would cause exclusion from assessment/reduction.
- Please follow the assignment guidelines. Failure to do so would cause exclusion from assessment and award of a reduction.
- Queries (if any) should be raised on **Piazza**. The Piazza link is: http://piazza.com/iit_delhi/fall2022/col333col671 and the access code is *col333*. Please keep track of Piazza posts. Any updates to the assignment statement will be discussed over Piazza. Please do not use email or message on Teams for assignment queries.

- **Please only submit work from your own efforts.** Do not look at or refer to code written by anyone else. You may discuss the problem, however the code implementation must be original. Discussion will not be grounds to justify software plagiarism. Please do not copy existing assignment solutions from the internet: your submission will be compared against them using plagiarism detection software. Copying and cheating will result in a penalty of at least -10 (absolute). The department and institute guidelines will apply. More severe penalties such as F-grade will follow.