SYSTEMATRIX

Interview Question Packet

You must complete ***at least* two** of the three questions within this packet to move onto the next stage of the interview process. Candidates who provide strong answers to all three of the questions will be considered top candidates moving into the next stage. However, candidates who provide outstanding answers to two questions can also be considered top candidates. Please submit the completed answers with a list of your available times for a phone interview over the next two weeks after receiving this packet to cameron@systematrixsolutions.net. Good luck!

1. Create a branch out Force-Directed Graph [*Visualization*]:
Using d3.js or a JavaScript visualization library of your choice, create an interactive force directed graph of character co-appearance in Les Misérables. However, instead of displaying the entire graph at once, start by just showing one node at the beginning (say, the character Marguerite) . Upon user action on that node (such as a double-click), the graph showed be updated to show her immediate neighbors (Fantine, Valjean). Clicking on Fantine now should show her own neighbors, and so on. Make sure to keep track of the nodes and links already in the network, so there are no duplicate nodes created on a click action. Further, any links between nodes just added to nodes already in the network must be shown. The end result of completely exhausted graph should have the same structure (nodes and links) as the graph here. Along with your code, please provide a brief write-up of your algorithm. Feel free to use any color scheme and aesthetics for the graph!
DATA AND GRAPH STRUCTURE: https://bl.ocks.org/mbostock/4062045

2. Find the most semantically similar sentence [*NLP*]
You have been asked to write an algorithm capable of comparing a given input sentence to a database of sentences, in order find the sentence within the database most similar to the input sentence. A colleague has already started the code, but has gotten stuck. Complete ***compare_sentences.py***, such that when given *input_sentence* and *corpus_file*, the script will return the sentence most similar to *input_sentence*. Two sentences are considered maximally similar in this context if the sentences are semantically equivalent. For instance, “I ate dinner after work.” and “I went out for food

once work was finished" are considered more similar than "I ate dinner after work." and "I skipped dinner after work."

In addition to completing the code, please answer the following questions:

1. Is there a more efficient way to handle the loading and comparison of the input sentence with the sentence database than your colleague has written in the code? If so, please explain and make the improvement to the code.
2. Why is the sentence vectorization method you have implemented ideal for this task?
3. Why is the vector comparison method you have implemented ideal for this task?
4. Are there any other improvements you made/could make to further improve the speed or accuracy of sentence comparison.

You are permitted to use open source Python libraries to solve this task. Also note that you can modify the existing code in *compare_sentences.py* as you see fit, so long as the script takes in a sentence as input and returns the most semantically similar sentence as output. **Use the attached *LEAVES OF GRASS* file as your corpus file.**

Example:

```
>> most_sim_sentence = find_similar_sentence('I sing the electric body', 'LEAVES OF GRASS')
>> print 'Most Similar Sentence:  ', most_sim_sentence
Most Similar Sentence:  I Sing the Body Electric
```

3. Design a search engine for faces [*Computer Vision*]

You have been asked to design a search engine for faces, so that a user can upload an image of a face and determine whether that face appears in database of known persons. The database currently consists of 5 million people and their respective faces. Please explain your approach to solving this problem in as much detail as possible, including pseudocode, diagrams, or actual code, if applicable.

a) How will you tell if two faces are the same or not?
b) Why did you choose the method you specified in (a) and how does that method compare to other available methods?
c) How will you ensure that your comparison of the input face against the database of faces is sufficiently fast? What sort of techniques would you implement to optimize the speed of determining whether a match exists or not?
d) How does the optimization technique you specified in (c) compare with other methods?