Coursework 2: Vector Space Semantics for Similarity between Friends Characters (include tips)

Due: Monday 8th January, 2024 (11.59pm GMT)

Instructions

This assignment is marked out of 100 and counts for 40% towards your final grade for the module.

Note: This is an individual coursework. You must attempt the questions yourself and provide strong evidence for understanding your method in both your notebooks and your report.

Objective: In this assignment, you will be creating a vector representation of a document containing lines spoken by a character in the Friends script data (i.e. from the training file), then improving that representation such that each character vector is maximally distinguished from the other character documents. This distinction is measured by how well a simple information retrieval classification method can select documents from validation and test data as belonging to the correct class of document (i.e. deciding which character spoke the lines by measuring the similarity of those character document vectors to those built in training). Much of the background for this part is in Lecture 8 and 9 on Semantics, though you should use all of your knowledge across the module.

How to submit: Follow the below instructions, and submit well documented code as **one or more IPython files** (.ipynb) building on the template file "NLP_Assignment_2_distributional_semantics.ipynb" as your starting point (Python 3.7+). You must **also submit a 2-page report** where you describe how you achieved each question briefly, with the relevant observations asked for below. Training, validation and test data are provided in three separate files ('training.csv', 'val.csv' and 'test.csv'). For this coursework you are only allowed to use a maximum of the first **300 lines** of each character in the training data 'training.csv' to create the training documents and a maximum of the first **30 lines** in the validation and test data (from 'val.csv' and 'test.csv').

Before submit your work, ensure your code runs from top to bottom without errors. If you do use more than one IPython file, it must be clear which file corresponds to which questions.

A simple vector representation for each character document is done for you to start with in this code, as is the pipeline of similarity-based information retrieval evaluation. You need to improve the character vector representations by pre-processing, feature extraction and transformation techniques, as per Questions 1-6 below, which you need to complete as instructed.

Question 1. Improve pre-processing (20 marks)

Using the pre-processing techniques you have learned in the module, improve the `pre_process()` function in the template, which currently just tokenizes text based on white space.

When developing, use the first 300 and 30 lines from the training and validation files. To check the improvements by using the different techniques, use the `compute_IR_evaluation_scores()` function provided in the template. The **mean rank** is the main metric you need to focus on improving throughout this assignment, where the target/best possible performance is **1** (i.e. all test/validation data character documents are closest to their corresponding training data character documents) and the worst is **10**. Initially, the code in the template achieves a mean rank of **4.0** and accuracy of **0.3** on the test set and a mean rank of **4.2** and accuracy of **0.1** on the validation set - you should be looking to improve those, particularly getting the mean rank as close to 1 as possible.

HINT: A minimum of **five** prepossessing techniques is recommended for this question.

Question 2. Improve linguistic feature extraction (30 marks)

Use the feature extraction techniques you have learned to improve the `to_feature_vector_dictionary()` and `create_document_matrix_from_corpus()` functions above. Examples of extra features could include extracting n-grams of different lengths and including POS tags. You could also use sentiment analysis or another text classifier's result when applied to the features for each character document. You could even use a gender classifier trained on the same data using the GENDER column **(but DO NOT USE the GENDER column directly in the features for the final vector)**.

Matrix transformation techniques like TF-IDF are popular options to be used to improve the performance in distributional semantics.

Other options include using feature selection/reduction with techniques like minimum/maximum document frequency and/or feature selection like k-best selection using different statistical tests.

Again, develop your system using the training and validation sets and note the effect/improvement in mean rank with the techniques you use.

HINT: A minimum of **three** techniques is recommended for this question, one of which should be **TF-IDF**.

Question 3. Add dialogue context and scene features (15 marks)

Adjust `create_character_document_from_dataframe()` and the other functions appropriately so the data incorporates the context of the line spoken by the characters in terms of the lines spoken by other characters in the same scene (before and after the target character's lines).

HINT: you should use the *Episode* and *Scene* columns to check which characters are in the same scene to decide whether to include their lines or not. Only the lines from the same *Scene* can be added as the context as lines from different *Scene* are irrelevant. **(but DO NOT USE the GENDER and CHARACTER columns directly).**

Question 4. Parameter Search (15 marks)

It is a good practice to conduct a systematic parameter search instead of a random search as this will give you more reliable results. Given the scope of this assignment, it is possible to conduct a **grid search** on options you decided to try within the individual questions. The grid search should be done within the individual questions (i.e. Q1-Q3), and the later question should adopt the best settings from the previous questions.

HINT: There is no need to do a grid search over all configurations from all questions as this will easily make the search unrealistic. E.g. Suppose we need 32, 90, and 4 runs to finish the grid search within questions, a cross-question grid search would need 32x90x4 = 11520 runs!

Question 5. Analyse the similarity results (10 marks)

From your system so far run on the training/validation sets, identify the heldout character vectors ranked closest to each character's training vector which are not the character themselves, and those furthest away, as displayed using the `plot_heat_map_similarity()` function. In your report, try to ascribe reasons why this is the case, particularly for those where there isn't a successful highest match between the target character in the training set and that character's vector in the heldout set yet.

HINT: Observations you could make include how their language use is similar, resulting in similar word or ngram features.

Question 6. Run on final test data (10 marks)

Test your best system using the code below to train on the training data (using the first 300 lines per character maximum) and do the final testing on the test file (using the first 30 lines per character maximum).

HINT: Make any necessary adjustments such that it runs in the same way as the training/testing regime you developed above - e.g. making sure any transformer objects are initialized before `create_document_matrix_from_corpus()` is called. Make sure your best system is left in the notebook and it is clear what the mean rank and accuracy of document selection are on the test data.