Embeddings

MCDA5511 Assignment #3

Due: Nov 2 before midnight

Set Up

In this assignment you will work with sentence embeddings, as well as get to know your classmates. The assignment is based on the following code repository: https://github.com/ansonyuu/matchmaking. In the homework, you will learn about model criticism by exploring the weaknesses and limitations of this model pipeline.

IMPORTANT: For this assignment, you will complete the set up individually in tutorial, but the homework will be done in groups as per usual.

Start by downloading the data and saving as *classmates.csv*:
https://docs.google.com/spreadsheets/d/1NuWKCbpBfad8OilTXssnwglUK0BMXTtMraaod_iMOZA/edit#gid=0

The file contains names and brief descriptions of interests from your classmates. For example, mine is

- Greg Kirczenow, "I enjoy being outdoors in nature, trail running and surfing."

Next familiarize yourself with the repo.

- Clone the repo to your GitHub account. Note that for the homework,
- Review the documentation for the various packages used in the repo to understand what the code does, particularly for the embedding and dimension reduction steps.
- While it is not standard practice to include data in a GitHub repository, in this case, the data file is small, and we want to be able to track changes made to it. Download *classmates.csv* to your local folder so that it will be captured in your repo when you push your changes.
- Since UMAP is a stochastic algorithm, you will need to set a seed to get the same results as your classmates. You can do this by replacing the line

  ```
  reducer = umap.UMAP()
  ```

  with

  ```
  reducer = umap.UMAP(random_state=42) # set the seed to 42
  ```
- Make sure you have your python environment and dependencies set up properly so that you can run the code from start to finish.
- Replace *sample.png* in your local folder with the *visualization.png* image your code generates from *classmates.csv* so that it will be displayed on your readme page in your main branch.
- Push the changes you have made to your repo main branch for safekeeping. Your repo main branch should now include the changes you made to the code above, as well as *classmates.csv*. Your readme should display your version of *sample.png* showing your classmates.

Now, discuss the results with your classmates.

- Looking at *visualization.png*, find one classmate who is close to you in embedding space, and one that is far away. With each classmate:
  - First, compare your results. If your results differ, try to find the source of the discrepancy.
  - Next, discuss reasons why you think the model placed you close/far in the embedding space.

Homework

For this assignment, you will use GitHub to submit your homework.

- If you wish, you can develop your own code base using packages of your choice. For example, you might prefer to work with https://github.com/koaning/embetter. If you choose to use packages of your choosing, you are responsible for ensuring that they have the same basic functionality as the original.
- **DO NOT use Jupyter notebooks for this assignment.** Instead, create a codebase using the best practices you learned in the tutorial, with an *environment.yml* file, a *main.py* file, and any other supporting files you need.
- Create new branches to run experiments as described in the questions below.
- Submit your repo for grading by inviting the instructors to collaborate on the project in GitHub when you are finished.
- Add code that saves the *person_embeddings* dictionary to a file in your local folder. You may use a file format of your choice – json for example – but use the file name *embeddings*. You will use this file later to examine the impact of model changes.

(1) Write a paragraph to explain the concept of word and sentence embeddings to a non-technical audience. Assume they do not know what a vector is but want to understand how embeddings capture the meaning of words. Feel free to include an image or two to illustrate your description and use examples from *classmates.csv*. Write less than 500 words. Save your paragraph in your *readme* in the main branch of your repo, in a subsection called "What Are Embeddings?" below *sample.png*.

Before proceeding, spend some time to clean up the code in your main branch. Make sure it is well-documented and follows the coding conventions you learned in tutorial.

We can think of this model pipeline as a machine learning model with three main components: the data, the embedding component, and the dimension reduction component. Let's explore each of these to determine how they impact the final output.

(2) Data: It is always important to consider how the idiosyncrasies of your dataset impact your results.
   - Create a new branch from the *main* branch called *data-analysis* so that you can perform the experiments described in this section without contaminating the *main* branch.
   - Pick three sentences from *classmates.csv* to modify. Make sure to make one major change and one minor change so that you can get a sense of the range of impacts. Examples of changes could include replacing a key word with a synonym, antonym, or homonym, or changing the phrasing of the sentence.
   - Re-run the code to generate a new embedding.
   - Create a new script called *data_comparison.py* that loads the old embeddings you saved previously and compares them to the ones you just generated. Compare the embeddings for the three modified sentences using a distance metric such as cosine similarity or dot product similarity.
   - Add a subsection to your main branch readme called "Data Analysis" explaining your results in less than 300 words. Start by noting what changes you made to the data to make things clear to the reader. Then explain whether the changes you made to the data had a large or small impacts on the embeddings and give a reason why.

(3) Embedding: The *sentence_transformers* package provides a variety of models to choose from - see https://www.sbert.net/docs/pretrained_models.html. Your main branch uses *all-MiniLM-L6-v2*. Investigate how a different model changes the results.

- Create a new branch from the *main* branch called *embedding-sensitivity-tests* so that you can perform the experiments described in this section without contaminating the *main* branch.
- Starting with the main branch code, switch to a different model of your choice and rerun your code.
- Create a new script called *model_comparison.py* that compares the embeddings produced by the two models. In this case, it doesn't make sense to use a similarity metric, since different models have different embedding spaces. Instead, we can use a measure of rank correlation such as Spearman's ρ to compare the order each model places your classmates, from closest to you to farthest. Compute the similarity between yourself and each classmate for both models, sort the results from closest to farthest, and compute the rank correlation between them.
- Add a subsection to your main branch readme called "Embedding Sensitivity Tests" explaining your results in less than 300 words. To what extent are your results sensitive to model choice? Your answer should include both quantitative considerations (e.g. rank correlation) and qualitative considerations (e.g. does a different embedding significantly change which classmates are close to you).

(4) Dimension reduction: Now let's turn our attention to the UMAP algorithm. This algorithm projects the 384-dimensional sentence embeddings to 2 dimensions so they can be displayed visually. For this question, the goal is to determine the extent to which the UMAP algorithm is adequate for our purpose of identifying classmates who have similar interests. For this exercise, use the *all-MiniLM-L6-v2* model.
   - Create a new branch from the *main* branch called *umap-tuning* so that you can perform the experiments described in this section without contaminating the *main* branch.
   - In the setup, we set a seed in the UMAP function to ensure the results are reproducible. Change the seed and rerun the code a few times, observing how *visualization.png* changes.
   - Hyperparameter tuning: Now, let's see if you can find hyperparameters such that UMAP faithfully reflects the similarity metric.
     - Review the UMAP documentation and take note of the several dozen parameters the UMAP method accepts. Determine which parameters you would like to tune, which you would like to set, and which to leave at default settings. For example, you may wish to tune `n_neighbours` and `min_dist`, and choose `metric` based on your knowledge of the problem at hand.
     - Use a hyperparameter search routine of your choice such as *hyperopt* or *optuna* to optimize the parameters you have chosen to tune.
     - For your optimization procedure, maximize average rank correlation between (a) cosine similarity in embedding space and (b) the Euclidean distance between points in the two-dimensional visualization, i.e. compute rankings for (a) and (b) relative to a given student and compute the spearman rank correlation coefficient ρ between them. Do this for all students, compute the average correlation, and use this metric to optimize the hyperparameters.
   - Once again, try changing the seed and rerunning your code a few times, but with the tuned version of your model. This time, observe changes in both *visualization.png* and the rank correlation metric you used to tune the model.
   - Add a subsection to your main branch readme called "Dimension Reduction Analysis" explaining your results in less than 300 words. Make sure to include the following in your explanation:
     - What does the sensitivity of *visualization.png* to the random seed indicate about the model stability of the original and tuned implementations?
     - Considering the tuned version of the model, assess the extent to which the UMAP algorithm captures the desired patterns in the data.

(5) Optional bonus question: Do you have any ideas for improvements to this model pipeline? If so, make the modifications and create a new branch called *model-improvements*. For example, you could use a different dimension reduction method such as tSNE and compare against UMAP. Add a section to your main branch readme called "Model Improvements" explaining what you did and why it is an improvement.

(6) Submit your repo for grading: Note that you can potentially share your repos from this course with prospective employers to demonstrate what you have done in school. As such, make sure that your main branch readme file clearly describes the project to someone who is seeing it for the first time. Spend some time cleaning up your repo to make it presentable.