Recommendations
MSc AI Coursework

Oana Cocarascu & Luca Grillotti

Problem description

In this coursework you will be analysing two types of recommendations. The first type of recommendation is related to route planning where you will be looking at London tube stations and compute the shortest path between two stations as well as how much a user would have to pay for his/her journeys. The second part is related to sentiment analysis. This can be seen as a sub-task in the decision making based on online reviews. As online reviews have become the chosen method of quality control for users in various domains, they now impact a customer's decision both positively and negatively.

To aid you in developing your program, you are supplied with a skeleton code.

Important:

- Do not modify any constructors.
- Do not import any other packages for **Part 1**.
- The files `tubemap.py`, `journey.py`, and `user.py` have example code of how to call and run the functions from each file. For example, to run `user.py`, you can execute the following command from the top (i.e. cloned) directory: `python3 -m tubemap.user`
- Do not modify the `main` function from the `main.py` file in **Part 2**.
- Make sure your code runs on the lab machines. You will lose marks otherwise.

How to get the files and how to hand in

You are provided with a git repository. You will use git to push your code changes to the repository. To get the code, type the following command:
`git clone git@gitlab.doc.ic.ac.uk:lab1920_autumn/cw2python_<login>.git`
replacing `<login>` with your Imperial username.

To submit on CATe, go to `https://teaching.doc.ic.ac.uk/labts` and click on the button `Submit to CATe` that is associated to the commit that contains the code you wish to submit.

Part 1

You are given a `json` file that contains data representing London tube map. Your task is to implement several functions related to loading and processing the data, determining the fastest route between two given stations, and computing the total price a user has to pay for his/her journeys.

Any journey that starts or ends between 06:30-09:30 and 17:00-20:00 is classed as peak. All other journeys are off-peak. If all of a customer's journeys on a given day are off-peak, their cap should be £7.00. If any of a customer's journeys are peak, their cap should be £8.80. Long journeys are over 10 stations and short journeys have fewer than or equal to 10 stations. The prices are as follows: Peak long: £3.80 ; Peak short: £2.90 ; Off-peak long: £2.60 ; Off-peak short: £1.90.

Implement the following functions in class `TubeMap`:

- `import_tube_map_from_json` which imports the json data from `london.json` and updates the attributes `graph_tube_map` and `set_zones_per_station`. Note that some stations belong to multiple zones (e.g. a station that has "zone": "2.5" should be recorded both in zone 2 and zone 3) **(10 marks)**
- `get_set_lines_for_station` which returns a set containing the names of the lines on which a station is found **(3 marks)**
- `get_set_all_stations_on_line` which returns the set of all the stations on a given line **(3 marks)**
- `get_fastest_path_between` which implements the Dijkstra algorithm to find the fastest path between two given stations (a link to the pseudocode of the algorithm is given in the file). You can use `float('inf')` to specify infinity. **(20 marks)**

Implement the following functions in class `Journey`:

- `is_list_successive_stations_valid` which returns `True` if all the stations given can be found in the tube map provided and all the pairs of successive stations are indeed connected **(2 marks)**
- `is_long_journey` which returns `True` if the journey is over 10 stations **(2 marks)**
- `is_on_peak` which returns `True` if the journey starts or ends during peak times **(2 marks)**
- `compute_price` which computes the price of the journey depending whether it is peak or not and if it is a long journey or a short journey **(2 marks)**

Implement the following functions in class `User`:

- `register_journey` which updates the attribute `journeys_per_date` given a journey's details **(3 marks)**
- `compute_price_per_day` which computes the price of all journeys in a given day **(3 marks)**

Part 2

You are given two files that contain positive and negative movie reviews. The data vector (X) and the labels (y) have already been created. Your task is to complete two classes that implement two classifiers: K Nearest Neighbours and a Feedforward Neural Network. You are allowed to use sklearn and keras libraries for this.

Implement the following functions in file models.py:

- split_train_test from class Classifier which should split X and y into training and testing **(2 marks)**
- train from class KNN which should:
  - create a pipeline that transforms the texts to TF-IDF features and uses a KNN classifier. Term Frequency–Inverse Document Frequency is a numerical statistic that reflects how important a word is to a text in a collection of texts. The steps in obtaining the TF-IDF are: first create a count vectorizer which takes a collection of text documents and converts it to a matrix of token counts and then normalizes it to obtain the TF-IDF representatiton. Hint: check the help page of sklearn.feature_extraction.text.TfidfVectorizer in sklearn to see how you can obtain these two steps **(5 marks)**
  - create a grid search to find the best number of neighbours and the best weight in a 10-fold cross-validation setting **(5 marks)**
  - train the model on the training data **(4 marks)**
- evaluate from class KNN which should compute the predictions for the testing data and return the classification report **(2 marks)**
- tokenize from class NN which tokenizes the training data keeping only VOCAB_SIZE words based on the word frequency and updates the data with the newly obtained vectors. Hint: You may find the function texts_to_matrix from keras.preprocessing.text.Tokenizer useful **(4 marks)**
- train from class NN which should:
  - create a feedforward neural network using the functional API, print the model summary, and save the network to a png file (make sure you push the file to git for the final submission) **(10 marks)**
  - train the model on the training data whilst specifying a validation split **(4 marks)**
- evaluate from class NN which should compute the predictions for the testing data and return the classification report **(5 marks)**
- predict_for_examples that overrides this method from the parent class **(5 marks)**

Implement the `knn_classifier` and the `nn_classifier` functions in file `main.py` which should create a classifier, train it, evaluate it (and print the classification report), and predict and print the classes for the examples provided in the file. **(4 marks - 2 for each function)**

How you will be marked

You will be assigned a mark according to:

- whether your program works or not;
- whether you have used meaningful names for variables;
- whether you have used a clear, appropriate and logical design.

You will not be marked on how good your classifiers are. However, your classifiers should have an accuracy above 70%.