User-centric Systems for Data Science **Prof. John Liagouris**
Fall Semester 2021 **liagos@bu.edu**

Assignment 3

Submission deadline: November 12 2021, 11:59 pm

This assignment requires building a simple Machine Learning pipeline for interpreting the predictions of a black-box medical image classifier. For this assignment, you will use (i) a popular Machine Learning library (PyTorch), and (ii) the state-of-the-art SHAP toolkit for model interpretability. The assignment is based on a real use case that is described below.

A group of researchers from the Department of Medicine at BU want to build an accurate MRI classifier that predicts whether a patient has Alzheimer's Disease (AD) or not (binary classification problem). The goal is to help doctors identify patients that are likely to develop AD in the future and suggest appropriate treatment. In this case, the input data are 3D MRI instances of high resolution (182x218x182). *The researchers have trained a Convolutional Neural Network (CNN) on a large private dataset that consists of MRIs from patients who have been diagnosed with AD at different stages and from people with normal cognition.*

The learned model has pretty good prediction accuracy, however, it is a black-box and the doctors cannot take its predictions into account in decision making without convincing explanations. In order to use it in clinical practice, doctors need an automated way to understand how the model generates its predictions. In particular, they want to see which areas of the brain the model focuses on and verify that these areas are consistent with clinical knowledge of AD. Your solution must provide meaningful visual explanations to individual classifications as well as insights into what the model has learned.

The data you will use for the assignment are from the Alzheimer Disease Neuroimaging Initiative (ADNI) and can be found here. This dataset contains a sample of 19 3D MRIs from real patients along with their segmented versions that define distinct regions of the human brain.

1. Data schema

The MRIs you will use are given in the form of Numpy arrays (.npy) as expected by the pretrained CNN model. The segmented MRIs are given as NIfTI files (.nii) that you can manipulate with the NiBabel library. Each MRI is associated with a set of metadata that are given in a CSV file (`ADNI.csv`). The schema includes 31 attributes (categorical, binary, and numerical). These are not cleaned data, that is, some values are missing from the CSV file, while others may be inconsistent or erroneous. The attribute (`AD`) contains the MRI labels (ground truth) that indicate whether the patient has (`AD=1`) or not (`AD=0`).

2. TASK I: Load model and data (credits: 10/100)

The first task is to load the pretrained CNN model in memory from the given checkpoint (`cnn_best.pth`) using PyTorch. At this step, you need to divide the 19 MRIs into two parts:

A. The first part contains the test data, i.e., the MRIs that you will use to probe the CNN model and generate classifications. Make sure that this dataset contains at least 5 MRIs.

B. The second part contains the background data, i.e., the MRIs that will be used by SHAP for perturbing the instances to approximate the Shapley values (see Lectures 12 and 14).

Report how many instances from your test dataset are classified correctly.

Hint #1: You may want to have a look at the PyTorch model and data loaders.

Hint #2: After implementing the whole pipeline, you can also experiment with different ratios of test and background data sizes to see if (and how) they affect the results.

3. Task II: Generate SHAP values for individual pixels (credits: 40/100)

The second task is to compute the SHAP values for the pixels of the MRIs whose predictions we want to explain. In this case, each pixel (element of the Numpy array) corresponds to a feature value that has a unique SHAP value for the given classification. The output of this task is another set of Numpy arrays (one for each input MRI) that contain the SHAP values.

Hint: You may use the DeepExplainer or the GradientExplainer from the SHAP repository.

4. Generate SHAP heatmaps on 2D MRI slices (credits: 10/100)

The third task is to extract 2D slices from the 3D MRIs and plot the SHAP values as heatmaps on the 2D images. Each heatmap serves as an explanation that highlights the individual pixels that contribute positively or negatively to a particular prediction, as shown in Fig. 1.

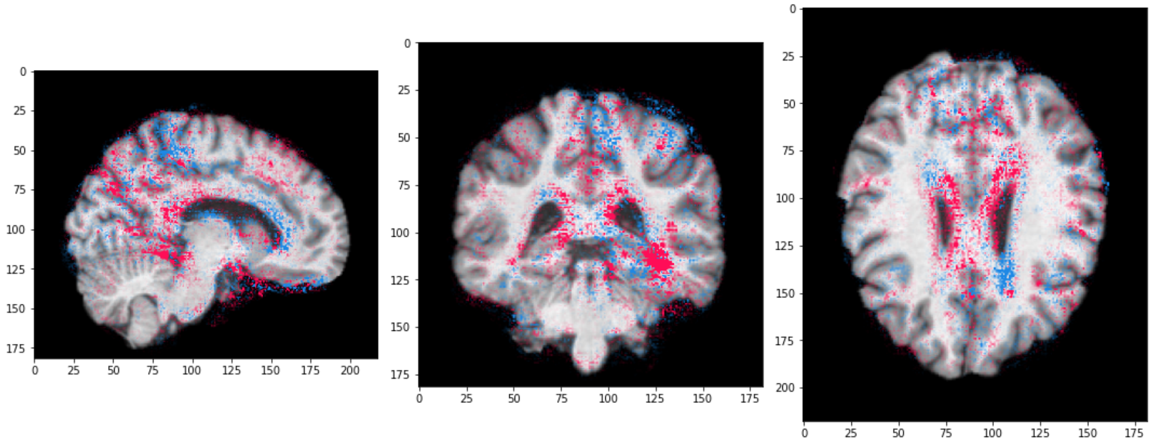

Fig. 1: Pixel-based SHAP heatmap on three 2D slices of a 3D MRI (side, back, and top views). In this example, red pixels contribute positively to the (correct) AD prediction whereas blue pixels contribute negatively.

You must generate heatmaps for two randomly selected MRI instances, one classified as "AD" and one classified as "Not AD". Make sure that the CNN model predictions for both MRIs are correct according to their label (in `ADNI.csv`).

Hint #1: You may find this plotting example in the SHAP repository helpful.

5. TASK IV: Identify the most contributing brain regions per class (credits: 40/100)

The fourth and final task is to use the segmented MRIs and the brain regions in `data_util.py` to aggregate the SHAP values per brain region and report the top-5 most contributing regions per prediction class (AD and Not AD) across all test instances. To do so, you need to:

A. Compute the average SHAP value for each region in each test MRI. Let $m^{r,i}$ be the average SHAP value for all pixels of region *r* with respect to the CNN prediction for the *i*-th MRI.

B. For each region *r*, compute the average $m_r = AVG_{\forall i}(m^{r,i})$ of all values from the previous step.

C. Output the top-5 list for each class in a CSV file.

An example of such a list should look as follows:

```
['TL hippocampus L', 'TL amygdala R', 'cerebellum R', 'OL lateral remainder occipital lobe R', 'thalamus L']
```

6. Testing

You must have a simple test for each operator you implement and we strongly recommend using Pytest for this purpose. In case you are not familiar with Pytest, you might want to first spend some time reading the code snippets provided in the documentation.

All test functions must be added to the separate `tests.py` file provided with the code skeleton. Before submitting your solution, make sure the command `pytest tests.py` runs all your test functions successfully.

7. Logging

Logging can save you hours when debugging your program, and you will find it extremely helpful as your codebase grows in size and complexity. Always use Python's logger to generate messages and avoid using `print()` for that purpose. Keep in mind that logging has some performance overhead even if set to INFO level.

8. Git

You will use git to maintain your codebase and submit your solutions. If you are not familiar with git, you might want to have a look here. Note that well-documented code is always easier to understand and grade, so please make sure your code is clean, readable, and has comments.

Make sure your git commits have meaningful messages. Messages like "*Commit*" or "*Fix*", etc. are pretty vague and must be avoided. Always try to briefly describe what each commit is about, e.g. "*Add DataLoader*", "*Fix bug in SHAP plotting function*", etc., so that you can easily search your git log if necessary.

Each time you finish a task (including the related tests), we strongly recommend that you use a commit message "Complete *Task X*". You will find these commits very helpful in case you want to rollback and create new branches in your repository.

9. Deliverables

Each submission must be marked with a commit message "*Submit Assignment X*" in git. If there are multiple submissions with the same message, we will only consider the last one before the deadline. In case all your submission commits are after the deadline, we will only consider the last commit, however, **late submissions will be eligible for up to 50% of the original grade**.

Your submission must contain:

1. The code you wrote to solve the assignment tasks (in `data_util.py` and `explain_pipeline.py`)

2. The code you wrote for testing (in `tests.py`)
3. The heatmaps and the top-5 lists you generated for Tasks III and IV

Before submitting your solution, always make sure your code passes all tests successfully.

10. Resources

- PyTorch: https://pytorch.org
- SHAP: https://github.com/slundberg/shap
- Python tutorial: https://docs.python.org/3/tutorial/
- Python logger: https://docs.python.org/3/library/logging.html
- Pytest: https://docs.pytest.org/en/stable/
- Git Handbook: https://guides.github.com/introduction/git-handbook/