Practical session

Deep emergency medical triage from free text dispatcher observations

Biomedical Data Science

Pablo Ferri Borredà

Biomedical Data Science Lab (BDSLab)

ITACA, UPV

Contents

1. Objective 3
2. Material 3
3. Evaluation 3
4. Tasks 3
4.1. Block I. Data preparation and preprocessing 3
4.1.1. Objective 3
4.1.2. Notes 4
4.1.3. Questions & exercises 4
4.2. Block II. Model training, selection and evaluation 4
4.2.1. Objective 4
4.2.2. Notes 5
4.2.3. Questions & exercises 5

1. Objective

To develop and evaluate a Deep learning model to aid non-clinical dispatchers to classify emergency medical call incidents by their life-threatening level (yes/no) based on free text fields.

2. Material

- Seminar:
  - PDF file: Deep ensemble multitask classification of emergency medical call incidents combining multimodal data
- Template code files:
  - dataprep.py
  - datapreproc.py
  - encoder.py
  - feeder.py
  - textmodel.py
  - textprepar.py
  - textpreproc.py
  - traineval.py
- Data file:
  - obs_lifethread.csv
- Metadata file:
  - abbrev2word_map.csv

3. Evaluation

This practical session will be evaluated taken into account the code files, which must be submitted in the Poliformat task.

4. Tasks

4.1. Block I. Data preparation and preprocessing

4.1.1. Objective

To carry out a proper preparation and preprocessing of your data, especially the free text fields, by the accomplishment of a set of subtasks.

4.1.2. Notes

- Use the classes and methods provided in the templates.
- Respect the structure of the classes and the working pipelines.

4.1.3. Questions & exercises

Using the script *dataprep.py* as your template:

B1.1. Load your data using the proper column delimiter.

B1.2. Use the method one_hot_encode of class OneHotEncoder to generate One Hot Encoding variables from the life-threatening label.

B1.3. Invoke the *prepare* method to prepare your 'OBSERVATIONS' of *TextPreparator* class by transforming sentences to lowercase, deleting punctuation marks, tokenizing words, and mapping abbreviations.

B1.4. Split your data using a holdout methodology, considering 80% of the data for training and the rest for testing.

B1.5. Observe the rest of the code in and identifying the lines where next steps are performed:

Map words to indexes	
Extract labels	
Convert numbers to tensors	
Generate datasets	
Configure batch sizes	
Generate dataloaders	

4.2. Block II. Model training, selection and evaluation

4.2.1. Objective

To gain experience with PyTorch, understanding which are the basic steps to define and train a Deep learning model in PyTorch. Study the influence of some hyperparameters in model performance.

4.2.2. Notes

- Use the classes and methods provided in the templates.
- Respect the structure of the classes and the working pipelines.
- Reuse code from prior practical sessions.

4.2.3. Questions & exercises

Using *textmodel.py*:

B2.1 Change the LeakyReLu activation function by a relu activation function. You may have to make a search about how a relu is defined in PyTorch.

Using *traineval.py*:

B2.2. Import your preprocessed data (data loaders).

B2.3 Run some experiments, changing your hyperparameters, such as learning rate, architecture or even your batch size (defined in *dataprepoc.py*) until you find a *good* configuration based on the cross entropy values and the figures obtained (you will need to complete that part too). You may need to make a search about model modules (embedding, recurrent, output) in order to set properly hyperparam values.

B2.4 Get some metrics like the area under curve, accuracy or macro F1-score (at least) to estimate model performance in the test set.