Re-Energize DR3:

Code Documentation and Description:

Disaster Response Framework Web Application

UCL Energy Institute

Islands Laboratory

15th February, 2022

Code Dependencies

This code uses Python 3.7.9 running after Miniconda3 environments. The code has been tested using both with a GPU NVIDIA Geforce RTX 2060 and without it (CPU only). The list of dependencies are described in the file *requirements.txt*. To install them using Anaconda/Miniconda, simply create and activate an environment:

```
conda env create --name <your_environment_name> --file=requirements.xml
conda activate <your_environment_name>
```

Alternatively, if you already have a conda environment created for the core framework described in the extended documentation, you can activate it and list your packages using:

```
conda activate <your_previous_environment_name>
conda list
```

to check whether your installed versions match with the dependencies.

Summary

This document describes the code* to run the web interface app for Disaster Response Framework in your local machine. This code uses the models learned from the core repository *NLP-Disaster*, and the data both from the 'Mulilingual Disaster Response Messages' (MDRM) dataset and the 'Unified Records from CEH and MET for Extreme Flooding in the UK' (UnifiSevere). The instructions to run and access locally the web application are explained briefly in the *README* file located in the root directory of this repository *NLP-Disaster*. The code is divided into two main steps:

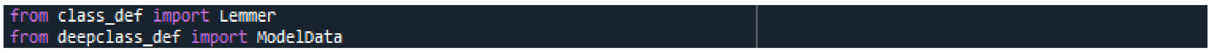
1. Loading data and models:
    a. Dependency classes.
    b. Benchmark datasets.
    c. Support Vector Classification and 'distilled BERT' deep learning models.

2. Templates rendering:
    a. Index or Master webpage.
    b. Multiclassification webpage.
    c. Deep classification webpage.

*https://github.com/islandslab/NLP-Disaster/tree/main/DisasterResponseApp

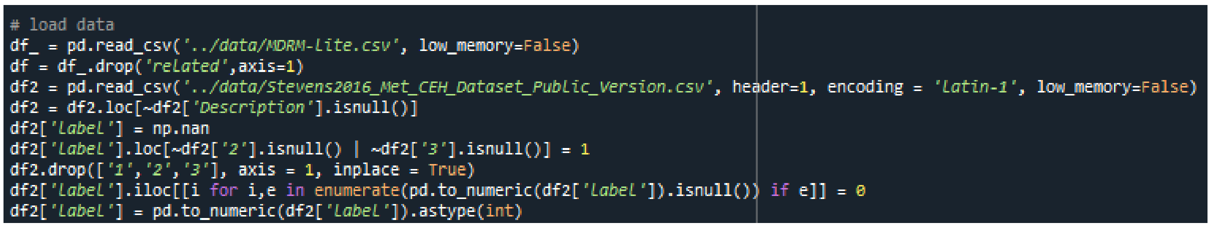
1. Loading data and models

Before loading the models, you need to **import the appropriate classes** from the Natural Language Processing Toolkit (NLTK) library for tokenization and lemmatizer, and the Transformers library to load our fine-tuned deep models.



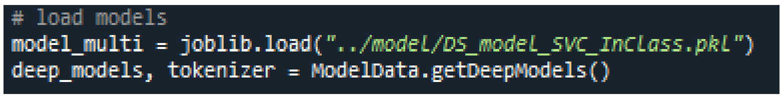
```
from class_def import Lemmer
from deepclass_def import ModelData
```

Then, you can **load the data** for both benchmark datasets using simplified CSV files that are converted into pandas dataframes.



```
# load data
df_ = pd.read_csv('../data/MDRM-lite.csv', low_memory=False)
df = df_.drop('related',axis=1)
df2 = pd.read_csv('../data/Stevens2016_Met_CEH_Dataset_Public_Version.csv', header=1, encoding = 'latin-1', low_memory=False)
df2 = df2.loc[~df2['Description'].isnull()]
df2['Label'] = np.nan
df2['Label'].loc[~df2['2'].isnull() | ~df2['3'].isnull()] = 1
df2.drop(['1','2','3'], axis = 1, inplace = True)
df2['Label'].iloc[[i for i,e in enumerate(pd.to_numeric(df2['Label']).isnull()) if e]] = 0
df2['Label'] = pd.to_numeric(df2['Label']).astype(int)
```

Next, you **load the** Support Vector multi-Classification model and the four 'distilBert' deep learning **models**:



```
# load models
model_multi = joblib.load("../model/DS_model_SVC_InClass.pkl")
deep_models, tokenizer = ModelData.getDeepModels()
```

Note that to load properly the data you need to make sure the model files exist in the specified paths here and in the script file `deepclass_def.py`. The deep models consist of big files which are not provided in the GitHub repository. Please refer to the main code documentation of the core framework for more details about training and loading the models.

Once you load the data and models, the next section describes how to render the templates for the web interface.

*https://github.com/islandslab/NLP-Disaster/tree/main/DisasterResponseApp

2. Templates Rendering

The template rendering uses the Flask library, which you should have been loaded at the beginning of the script:

```
from flask import Flask
from flask import render_template, request
```

2.a) Index or Master webpage

The first route for the app to go from the root is the *index*, which runs the code from the function `index()` in the script. This function will dump the dataframes loaded into their corresponding variables that will be used to create the five graphs to plot and render in the template.

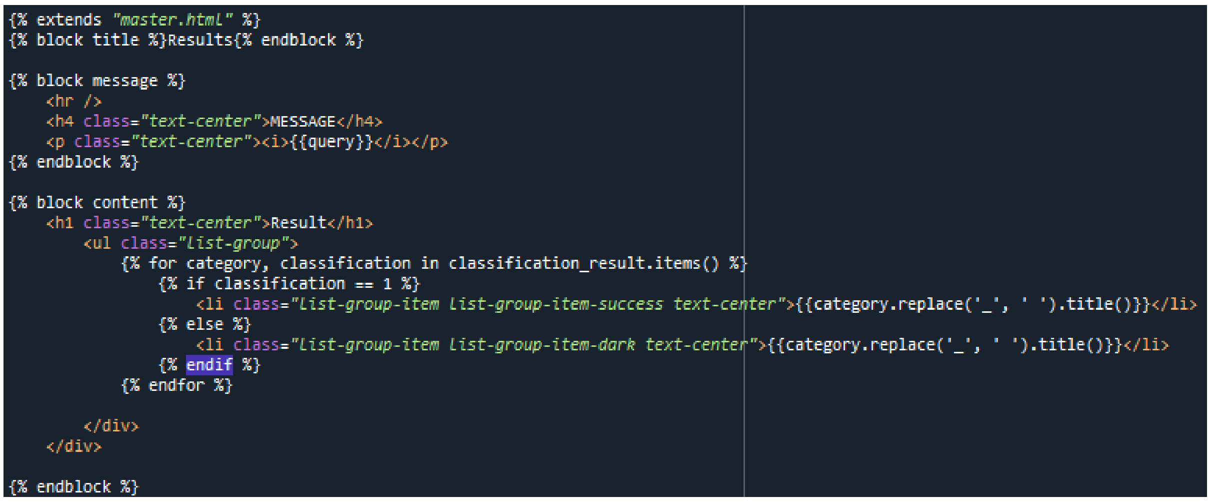
```
@app.route('/')
@app.route('/index')
def index():

    # assign data needed for visualisations and create graphs
    ## Disaster category
    label_sums = pd.Series({'Disaster': df_.iloc[:, 4].sum(), 'Non Disaster': len(df_)-df_.iloc[:, 4].sum()})
    label_names = list(label_sums.index)
    graphs = [{'data': [Bar(x=label_names, y=label_sums)],
               'layout': {'title': {'text':'Distribution of Disaster Category'},
                          'yaxis': {'title': "Count"},'xaxis': {'title': "Category"}}}]
    ## genre groups
    genre_counts = df.groupby('genre').count()['message']
    genre_names = list(genre_counts.index)
    graphs.append({'data': [Bar(x=genre_names, y=genre_counts)],
                   'layout': {'title': 'Distribution of Message Genres',
                              'yaxis': {'title': "Count"},'xaxis': {'title': "Genre"}}})
    ## and all categories
    label_sums = df.iloc[:, 4:].sum()
    label_names = list(label_sums.index)
    graphs.append({'data': [Pie(labels=label_names, values=label_sums)],
                   'layout': {'title': {'text':'Distribution of Disaster Categories','y': 0.95},
                              'yaxis': {'title': "Count"},'xaxis': {'title': "Category"}}})
    ## humanitarian standard and medical categories
    label_sums = pd.Series({'Humanitarian Standards': df.iloc[:, [6, 8, 20, 26, 28]].sum().sum(),
                  'Medical': df.iloc[:, [7, 12, 13, 14]].sum().sum()})
    label_names = list(label_sums.index)
    graphs.append({'data': [Pie(labels=label_names, values=label_sums)],
                   'layout': {'title': {'text':'Distribution of Humanitarian Standards and Medical Categories'},
                              'yaxis': {'title': "Count"},'xaxis': {'title': "Category"}}})
    ## Severity category
    label_sums = pd.Series({'Severe': df2.iloc[:, 22].sum(), 'Mild': len(df2)-df2.iloc[:, 22].sum()})
    label_names = list(label_sums.index)
    graphs.append({'data': [Bar(x=label_names, y=label_sums)],
               'layout': {'title': {'text':'Distribution of Severity Category from the "UnifiedCEHMET" dataset'},
                          'yaxis': {'title': "Count"},'xaxis': {'title': "Category"}}})

    # encode plotly graphs in JSON
    ids = ["graph-{}".format(i) for i, _ in enumerate(graphs)]
    graphJSON = json.dumps(graphs, cls=plotly.utils.PlotlyJSONEncoder)

    # render web page with plotly graphs
    return render_template('master.html', ids=ids, graphJSON=graphJSON)
```

The created graphs will show the main statistics and categories for the two benchmark datasets that we use to learn the models in the core framework. These graphs are dumped to JSON variables to render the `master.html` file. This file contains HTML code to 1) load two forms for multiclassification and deep classification, respectively, and 2) load a javascript code to render the graphs using the `Plotly` library.

*https://github.com/islandslab/NLP-Disaster/tree/main/DisasterResponseApp

2.b) Multiclassification webpage

The first form of the `master.html` renders an extended template `multi.html` to show the results of the multiclassification for the given free text input from the text box in the form. These results are shown in the form of replicating the input text message, and showing the list of categories which the message has been classified (in green colour):



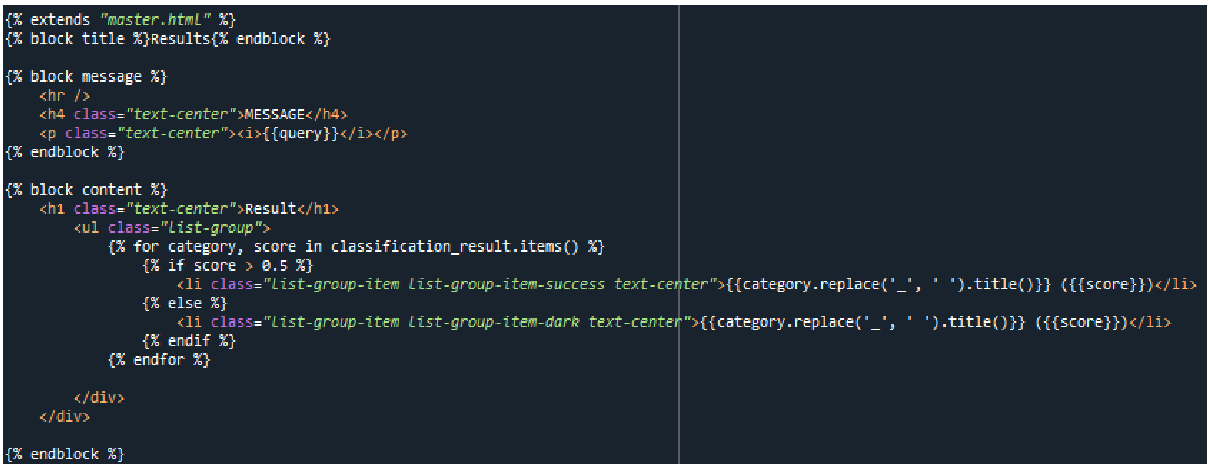
```
{% extends "master.html" %}
{% block title %}Results{% endblock %}

{% block message %}
    <hr />
    <h4 class="text-center">MESSAGE</h4>
    <p class="text-center"><i>{{query}}</i></p>
{% endblock %}

{% block content %}
    <h1 class="text-center">Result</h1>
        <ul class="list-group">
            {% for category, classification in classification_result.items() %}
                {% if classification == 1 %}
                    <li class="list-group-item list-group-item-success text-center">{{category.replace('_', ' ').title()}}</li>
                {% else %}
                    <li class="list-group-item list-group-item-dark text-center">{{category.replace('_', ' ').title()}}</li>
                {% endif %}
            {% endfor %}

        </div>
    </div>

{% endblock %}
```

The results of this multiclassification are calculated in the second route named *multi* for the app to go. This route runs the function `goMulti()` in the script, which retrieves the input text from the form as a `query` and classifies it into the multiple categories from the MDRM dataset. The classification outputs are dumped into a dictionary to be rendered in the `multi.html` file.

2.c) Deep classification webpage

The second form of the `master.html` renders another extended template `deep.html` to show the results of the deep classification for the given free text input from the text box in the form. These results are shown in the form of replicating the input text message, and showing the categories which the message has been classified for each model along with their confidence scores. These categories are shown in green colour if the confidence scores for the message to fall into these categories in each model are above 0.5.

*https://github.com/islandslab/NLP-Disaster/tree/main/DisasterResponseApp



```
{% extends "master.html" %}
{% block title %}Results{% endblock %}

{% block message %}
    <hr />
    <h4 class="text-center">MESSAGE</h4>
    <p class="text-center"><i>{{query}}</i></p>
{% endblock %}

{% block content %}
    <h1 class="text-center">Result</h1>
        <ul class="list-group">
            {% for category, score in classification_result.items() %}
                {% if score > 0.5 %}
                    <li class="list-group-item list-group-item-success text-center">{{category.replace('_', ' ').title()}} ({{score}})</li>
                {% else %}
                    <li class="list-group-item list-group-item-dark text-center">{{category.replace('_', ' ').title()}} ({{score}})</li>
                {% endif %}
            {% endfor %}

        </div>
    </div>

{% endblock %}
```

The results of this deep classification are calculated in the third route named *deep* for the app to go. This route runs the function `goDeep()` in the script, which retrieves the input text from the form as a `query` and classifies it using different models for information related to disasters, humanitarian, medical, or severe events. These models are based both on the MDRM and the UnifiSevere datasets. The confidence scores of the classification are calculated using your CPU without the need of dealing with GPU tensors – i.e. without the need of loading the torch library (uncommented at the beginning of the script). This simplifies considerably the calculation, avoiding the GPU dependency for the deployment of the web interface app. Similarly, the classification outputs are dumped into a dictionary to be rendered in the `deep.html` file.

*https://github.com/islandslab/NLP-Disaster/tree/main/DisasterResponseApp