nlp_exercise

This project is a proposal of a solution for the exercise proposed in `NLP_exercise.pdf`. In short, the exercise provides a dataset of tweets labeled as 1 or 0, where 1 indicates that the tweet seems to be a pro-ISIS one and 0 indicates that it doesn't. The goal is to build a solution able to take a tweet and retrieve the probability that it is a pro-ISIS one.

Preparing the environment

For the solution of this exercise I have used a python environment that can be recreated as a conda environment by following the next steps.

The virtual environment can be recreated by running this script script (assuming that we are at the root of the project).

```
$ cd scripts
$ ./create_conda_env.sh
```

The name of the resulting environment will be *nlp_exercise*. Once it has been created, other two additional dependencies must be installed on it. To install them, the environment must be activated.

```
$ conda activate nlp_exercise
```

The first additional dependency is FastText. It will be used to do language identification. It can be installed by running the script `scripts/install_fasttext.sh`.

```
$ ./install_fasttext.sh
```

The last dependency is NLTK stopwords. It can be installed by running the script `scripts/install_stopwords.sh`.

```
$ python install_stopwords.py
```

With all those dependencies installed, and the environment activated, the different scripts and notebooks in the exercise can be run.

Structure of the project

```
nlp_exercise
|   README.md
```

```
|   conda.yaml
|
└───scripts
|       ...
|
└───config
|       config.lr01.json
|       config.mlp01.json
|
└───data
|    |   Tweets.xlsx
|    |   Tweets.csv
|    |
|    └────dataset
|             balanced_dataset.csv
|             training_set.csv
|             validation_set.csv
|             testing_set.csv
|             ...
|
└───it
|       test.01.sh
|       test.02.sh
|       test.03.sh
|       test.04.sh
|       test.05.sh
|
└───models
|       modellr01
|       modelmlp01
|
└───notebooks
|       dataset_analysis.ipynb
|       model_evaluation.ipynb
|
└───src/main/python
        models.py
        train.py
        predict.py
        prepare_dataset.py
```

- `src/main/python` contains the main code of the project:
  - `prepare_dataset.py` is a script that takes an original dataset, and splits it into the training set, validation set, and testing set.
  - `train.py` is a script to launch the training of a new model.
  - `predict.py` is a script that loads a previously trained model and predicts over new text.
  - `model.py` is the library in which the majority of the code has been written. The rest of the scripts and notebooks just use the classes defined here.
- `data` contains all the datasets used and created in the project.

- `it` contains a bunch of integration tests. They may be especially interesting because contain examples of execution of the code in `src/main/python`.
- `config` contains the different configuration settings that were used to train different models
- `model` is the folder where trained models have been saved
- `notebooks` contains two notebooks with the parts of the exercise that require visualization of plots
- `scripts` contains several scripts that can be run to create a conda virtual environment with all dependencies needed to run the different scripts and notebooks in the project.

Execution of the scripts

In this project, I have trained two different models that can be used to take a tweet and predict if it is a pro-ISIS one or it isn't. The models have been saved in:

- `models/modellr01`
- `models/modelmlp01`

To train the models I have taken the original dataset `data/Tweets.csv` and I have preprocessed it in a specific way. The result of that preprocessing was saved in the folder `data/dataset`.

I have also used the trained models to predict over new text and evaluate the models.

Preparation of the dataset

The preparation of the dataset has been done by executing the following commands (we assume that we are in the root of the project).

```
$ cd src/main/python
$ python prepare_dataset.py -d ../../../data/Tweets.csv -s ',' -t 'Tweet' -
l 'ISIS Flag' -p 0.80 -o ../../../data/dataset/
```

The idea is that we take as input the original dataset `data/Tweets.csv` (parameter `-d`), the delimiter of fields of that CSV file is , (parameter `-s`), the field of the CSV where texts must be found is Tweet (parameter `-t`), the field where labels must be found is ISIS Flag (parameter `-l`), the proportion of data to be used in the training set is 0.80 (parameter `-p`), and the folder where resulting datasets will be saved is `data/dataset/`.

The result is a set of files saved in `data/dataset/` with a balanced version of the original dataset and the splits of that balanced dataset in training, validation, and testing sets (with a proportion of data of 80%, 10%, and 10% respectively).

Notice that we have an example of this command in `it/test.01.sh`.

Training of the models

Assuming that we are in the root of the project, to train the `models/modellr01` model I have run the follwing commnds:

```
$ cd ../src/main/python/.
$ python train.py -d ../../../data/dataset/training_set.csv -v
../../../data/dataset/validation_set.csv -s ',' -t 'Tweet' -l 'ISIS Flag' -
o ../../../models/modellr01 -c ../../../config/config.lr01.json
```

The idea is to train a model according to the configuration that was saved in the file `config/config.01lr.json` (parameter `-c`). That file contains the configuration needed to train just a logistic regression model. The training data to be used is `data/dataset/training_set.csv` (parameter `-d`), and the validation set is `data/dataset/validation_set.csv` (parameter `-v`). The resulting model will be saved in `models/modellr01`.

The second model that has been trained is a multilayer perceptron. It has been trained and evaluated with the same datasets, but it used a different configuration (parameter `-c`) and was saved in a diferent folder (parameter `-o`).

```
$ cd ../src/main/python/.
$ python train.py -d ../../../data/dataset/training_set.csv -v
../../../data/dataset/validation_set.csv -s ',' -t 'Tweet' -l 'ISIS Flag' -
o ../../../models/modelmlp01 -c ../../../config/config.mlp01.json
```

Notice that scripts `it/test.02.sh` and `it/test.03.sh` contain those commands.

Predicting over new text

Once the models have been trained, they can be used to predict over new text. Specifically, the testing set `data/dataset/testing_set.csv` previously created can be used to test our models.

With the following commands the model `models/modellr01` can be loaded and used to predict over that dataset:

```
cd ../src/main/python/.
python predict.py -d ../../../data/dataset/testing_set.csv -s ',' -t Tweet
-l Predictions -m ../../../models/model_lr01 -o
../../../data/dataset/predictions_lr01.csv
```

The output of the process is a new dataset `data/dataset/predictions_lr01.csv` with the texts and labels of the input dataset plus a new column Predictions with the scores predicted by the model.

The corresponding execution with the model models/modelmlp01 is as follows:

```
cd ../src/main/python/.
python predict.py -d ../../../data/dataset/testing_set.csv -s ',' -t Tweet
-l Predictions -m ../../../models/model_lr01 -o
../../../data/dataset/predictions_lr01.csv
```

Those executions can be found also in the scripts `it/test.04.sh` and `it/test.05.sh`.

Analysis

The dataset was originally provided in XLSX format (`data/Tweets.xlsx`). I exported it to raw CSV format, a more handy format to be processed by code: `data/Tweets.csv`.

I've done an exploratory analysis of the dataset. The code used to do that exploratory analysis can be found in the notebook `notebooks/dataset_analysis.csv`. It has been prepared to be self-explanatory.

The analysis followed these steps:

- Checking the size of the dataset.
- Checking the consistency of the data model in the dataset.
- Checking if there are many duplicates
- Checking if the dataset is balanced.
- Checking the nature and quality of the text.
- Checking the content of the texts.

The conclusions of the exploratory analysis have been the following:

- The dataset is highly unbalanced, but there may be enough instances of the positive class to undersample the negative one and train a model with a balanced version of the dataset.
- Texts need certain cleaning and normalization. It has UTF-8 encoding but some Unicode characters and line breaks appear escaped.
- Some characters cannot be correctly encoded in UTF-8.
- No duplicates were found.
- There are texts in different languages, but the vast majority of them are in English.
- Positive texts seem to have very characteristic words and phrases: *ISIS*, *#isis*, *Syria*, *#syria*, etc.
- Moreover, there are very few negative texts with those words: *ISIS* appears only in 32 negative texts 32, *#isis* only in 3, *Syria* in 19 or *#syria* 9.

Solution

Given the conclusions of the exploratory analysis of the dataset, I have decided to follow the approach that I explain in this section.

Unbalanced dataset

To tackle the problem of an unbalanced dataset, different strategies can be followed: undersampling the most populated class, oversampling the less populated, trying a weighted loss function, or trying algorithms more robust with unbalanced datasets such as Random Forest, try to model the problem as an anomaly detection one...

The easiest solution is undersampling the most populated class. For that, you need to have enough instances of the less populated class. In this case, we have ~17000 instances of the positive class. A resulting balanced dataset would have ~33000 instances. It may be enough to train a good binary classifier, at least as a baseline version. So I've decided to undersample the negative class and use a balanced version of the dataset to train my models. To make sure that the resulting selection of instances of the negative class is not biased, I've randomly shuffled the instances before I select the first ~17000.

That undersampling has been done with the `src/main/python/prepare_dataset.py` script.

Very few against-ISIS tweets

The fact that the most characteristic words in positive texts appear very little in negative texts suggests that there are very few against-ISIS tweets in the dataset. This would be a problem. It may be feasible to train a model able to identify tweets dealing with ISIS, but distinguish between pro and against ISIS tweets would be harder.

At least for the first version, I will skip the problem. We will see in the evaluation if this is actually a problem, and, if so, I will suggest an approach to solve it.

Texts in different languages

In the dataset, there are texts in different languages. However, the vast majority are in English (only ~4000 are in other languages). So I've decided not to give special treatment to non-English texts in the first version.

Preprocessing: cleaning and tokenization

To clean, normalize and tokenize the text I have written two classes in the `src/main/python/models.py` library:

- `TextNormalizer`: intended to:
    - Unescape escaped Unicode characters
    - Remove line breaks
    - Remove non-UTF-8 characters
- `TextTokenizer`: intended to:
    - Split text into tokens where specific Twitter tokens such as hashtags and user names are preserved as tokens.
    - Filter unwanted tokens through regular expressions and a stopwords list.
    - Lower case tokens.

Vectorization of the texts as Bag of Words

The text must be vectorized in a way that captures characteristic features of the pro-ISIS texts. I have seen in the exploratory analysis of the dataset that pro-ISIS texts tend to have very specific words and phrases. So maybe a representation of texts as a Bag of Words is enough to capture the characteristic features of those texts. No context-sensitive approaches such as RNN or encoders based on attention (transformers) seem to be needed, at least for a first version.

The BoW is calculated for pre-tokenized texts through a Keras Tokenizer object.

Multilayer Perceptron or Logistic Regression

I consider that we have enough data to train simple neural networks. So I decided to create in `src/main/python/models.py` a class `BinaryTextClassifierTrainer` that lets you train more or

less complex Neural Networks according to a specific configuration file. Then, the idea is to train different models with different configurations and decide what model works better on an empirical basis:

- I've trained a first model `models/modellr01` according to the configuration of `config/config.lr01.json`. Given that configuration, the model trained is, in fact, a simple logistic regression one, without hidden layers, with just an output neuron.
- I've also trained a model models/modelmlp01 that has been configured according to the configuration of `config/config.mlp01.json`. Given that configuration, it is a multilayer perceptron with a hidden layer of 250 neurons and the needed output layer with a single neuron.

Anyway, the output layer is always a one-neuron with a sigmoid activation function, as needed in binary classification problems. This function projects the logits of the output layer neuron in a range between 1 and 0.

`BinaryTextClassifierTrainer` uses Tensorflow to create and train the models.

Evaluation

The models `models/modellr01` and `models/modelmlp01` were trained by adjusting their hyperparameters according to their performance observed in the validation set. The resulting models have then been evaluated according to the testing set. That process of evaluation can be seen in the notebook `notebooks/model_evaluation.csv`. I also show the results here:

The metrics used, as usually for this kind of classification task:

- Precision
- Recall
- F1 score

The dataset used is `data/dataset/testing_set.csv`.

- It contains ~3300 texts (10% of the whole dataset)
- It doesn't contain bias because their thexts were selected randomly from the whole dataset

The results are the following:

Metrics	Logistic Regression	Multilayer Perceptron
Precision	0.964	0.963
Recall	0.906	0.910
F1 score	0.934	0.936

The conclusions about the performance of the model depend on the requirements of the specific use case in which the model would be used. In a specific use case, the recall score, for example, maybe not be high enough, because finding almost all pro-ISIS tweets is critical. But in general, we can say the performance seems relatively good. Notice that, given that the dataset used to train and evaluate are well-balanced, we can be confident about metrics like precision.

Caveats. As we feared before (see section *Very few against-ISIS tweets*), the models don't perform well with against-ISIS tweets. That's due to the fact that the vast majority of non-pro-ISIS tweets just deal with other topics. You can see examples in `notebooks/model_evaluation.csv`.

Next steps

- Including multilingual support. We could automatically translate non-English texts (maybe using several Huggingface open models). In that case, a previous language identification should be done, maybe with the FastText language identifier. Another approach could be using a multilingual encoder (multilingual BERT, Sentence Transformers, Universal Sentence Encoder) to represent text.
- Performing well with against-ISIS tweets.
    - As a first approach, I suggest using our current models to identify negative texts in the dataset that our models classify as positive. Those tweets would be candidates to be against-ISIS tweets. I suggest refactoring the balanced dataset by adding those tweets. I don't think that the resulting dataset would be significantly unbalanced. We should retrain the models with this dataset.
    - As a more serious approach, of course, we should oversample those against-ISIS tweets, ideally by collecting data.
    - As an imaginative approach, I would use our current models to identify tweets dealing with ISIS. And, as a second step, we could apply a sentiment analysis solution (ideally aspect-based sentiment analysis) over those tweets.