Exercise 1

The goal of this exercise is to build text classifiers using traditional machine learning techniques, which are precursors of current transformer-based classification approaches. In particular, we will build classifiers that learn to distinguish texts according to whether they contain conspiratorial thinking or critical thinking. The goal is to better understand the process of text classification and to become familiar with the data that you will use for the shared task: https://github.com/dkorenci/pan-clef-2024-oppositional/.

Description

Step 1: Loading the data

The provided scripts take care of loading the data into a dataframe with the following format:

```
      id                              text  label
0   5206  THIS IS MASSIVE Australian Sen...      1
1   1387  ‟ I ’m deeply concerned that t...      0
2  13116  2021 : They wanted to know you...      0
3  11439  Anthony Fauci once again defen...      0
4     98  Proof has emerged showing that...      0
```

Where `id` is the document id (i.e. the telegram message), `text` contains the full text of the message, and `label` contains the annotation of whether the text is an instance of conspiratorial thinking (label `1`) or critical thinking (label `0`).

Step 2: Text preprocessing

The goal is to build a text classifier that is able to predict whether a text is an instance of `conspiracy` or `critical` thinking. Traditional text classifiers are based on the bag-of-words assumption, according to which a text is a collection of words in which their position in the text does not matter. The preprocessing prepares the text to be fed into the classifier. This step can have a big impact on the performance of the classifier.

There are many possible preprocessing options. Some typical preprocessing steps are:

- Lower-casing: lower-case the text.
- Punctuation removal: removing the punctuation.
- Stop-word removal: removing stop words (i.e. common uninformative words like `'the'`, `'with'`, `'is'`).
- Lemmatization: converting all words to their lemma form (i.e. dictionary form)
- Part-of-speech filtering: keeping only words belonging to specific part-of-speech categories (e.g. noun, verb, etc.)

The `spacy` library provides powerful state-of-the-art natural language processing functionalities.

Splitting the dataset into training, development and test

Supervised learning consists of training and testing a model. A model is built using the training data and its performance is consequently evaluated on unseen examples. The development set is used for testing during the experiments, evaluation on the test set should be done at the very end.

Vectorising the data

Classifiers work with texts represented as vectors. These vectors should represent the meaning of the text. The question of how to represent meaning computationally is at the core of NLP, and has evolved considerably from the first bag-of-words approaches to today's transformer-based embeddings.

The bag-of-words approach assumes that a document can be represented through the frequencies of its words, regardless of the position of the word in the document. Each word in the preprocessed document collection is a feature: this step converts the dataset into a document-term matrix. This kind of vectorizing is called sparse (documents are represented with very long vectors with mostly zeros). This approach is a precursor to static word embeddings (like word2vec representations, see chapter 6 in https://web.stanford.edu/~jurafsky/slp3/ for more information) and contextualised embeddings (like BERT or GPT, see chapter 11 in https://web.stanford.edu/~jurafsky/slp3/ for more information).

The `scikit-learn` library (https://scikit-learn.org/stable/) provides two convenient functions for vectorizing text: `CountVectorizer` and `TfidfVectorizer`. The first converts a collection of text documents to a matrix of token counts. The second converts a collection of text documents to a matrix of tf-idf-weighted tokens.

Choosing a learning algorithm and training a classifier

The next step is to select the classification algorithm that will learn the relation between features and labels. The `scikit-learn` library provides many different classification algorithms. They take as input the (vectorized) training data and learn to predict the label of a new (also vectorized) text.

Evaluating and reporting performance

Common performance metrics are precision, recall and f1-score. MCC (Matthews correlation coefficient) is a recommended metric for binary classification. Again, the `scikit-learn` library provides implementations for all these metrics.

Instructions

1. Setting up a `conda` environment is highly recommended. You can use the same environment as the one you will need for the oppositional thinking shared task:

https://github.com/dkorenci/pan-clef-2024-oppositional. Python 3.10 is recommended.
2. Install the following libraries: `scikit-learn`, `pandas`, `numpy`, `spacy` and `gensim`.
3. Install SpaCy models for English and Spanish:

```
python -m spacy download en_core_web_sm
python -m spacy download es_core_news_sm
```

4. Download the dataset (`dataset_oppositional`) and the two python scripts (`exercise_1.py` and `utils.py`) and put them in the same directory. The directory should look like this:

```
├── dataset_oppositional/
│   ├── dataset_en_train.json
│   └── dataset_es_train.json
├── exercise_1.py
└── utils.py
```

5. Have a look at `exercise_1.py` and run it.
6. Go to `utils.py`, there are several chunks of code you will need to fill. Search for the keyword `TODO`. The exercises are described in the section below, and also in comments in the code.
7. You will only need to submit the `utils.py` file (exercises 1 to 4) and a word/pdf/txt file with the answers to the questions in exercise 5.

Exercises

- **Exercise 1:**
  - In `exercise_1.py`, change the preprocessing approach in line 44 from `"basic"` to `"spacy"`.
  - In `utils.py`, search for `“TODO Exercise 1”` and implement the `"spacy"` preprocessing approach according to the instructions provided in the comment. For information on how to get started, see the following tutorial: https://spacy.io/usage/spacy-101. Information on the class `Token` attributes will also be useful: https://spacy.io/api/token#attributes.
- **Exercise 2:**
  - In `exercise_1.py`, change the preprocessing approach in line 44 to `"spacy_pos"`.
  - In `utils.py`, search for `“TODO Exercise 2”` and implement the `"spacy_pos"` preprocessing approach according to the instructions provided in the comment.
- **Exercise 3:**

    - In `exercise_1.py`, change the classification approach in line 70 from `"clf1"` to `"clf2"`. In `utils.py`, search for `“TODO Exercise 3”` and do the following changes based on the `"clf1"` approach:
        - Change the vectorizer from a count vectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) to a Tf-Idf vectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html). See more detailed instructions in the comment.
        - Use logistic regression to predict the classes (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression), instead of multinomial Naive Bayes (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html). Optionally, you can experiment with different choices of parameters.
- **Exercise 4 [Optional]:**
    - In `exercise_1.py`, change the classification approach in line 70 to `"w2v"`.
    - In `utils.py`, search for `“TODO Exercise 4”` and change the vector representation to averaged word embeddings. This means that, instead of representing the document as a sparse vector, it will be represented with a dense vector. To do so, we will find the embedding of each word in the document, and average the embeddings. Therefore, all documents will be represented as vectors with the same size (i.e. the size of the word embeddings). Averaging the word embeddings of a document is a common (even if naive) approach to representing the document. You can download the embeddings from FastText: https://fasttext.cc/docs/en/crawl-vectors.html#models. Note that it takes a while to load the file. More detailed instructions in `utils.py`.
- **Exercise 5: Answer the following questions:**
    - Report a table with the results of different combinations (languages, data processing approaches, classifier approaches) and write a paragraph discussing the results and describing your observations.
    - Point out at least two advantages and two problems of the bag-of-words approach. Can you think of any cases in which it could be useful to use these kinds of approaches today?

Bibliography

- D. Jurafsky, J. H. Martin. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 3rd edition, 2023: https://web.stanford.edu/~jurafsky/slp3/. Chapters 4, 5 and 6.
- C.D. Manning, P. Raghavan and H. Schuetze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265.
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).