| The code includes | Default data directory | Required dependencies | To run the code under the default setting |
|---|---|---|---|
| 1- main.py<br>2- data_prepare.py<br>3- data_tokenizer.py<br>4- model.py | /data/input/test<br>/data/input/train<br>/data/output/ | are available as a Conda environment<br><br>environment.yml | >> python3 main.py |

Running the code you will be provided by a summary of what happens at each step as follows

* Reading train data: number of loaded entries, number of positive and negative entries, the 10-top most frequent elements, address of the file to which all words and their frequency are saved, etc.
* Train and test results of the SVM model (loss, accuracy and F1)
* Train and test results of the NN model (loss, accuracy and F1)
* A chance to try the both models by entering your sentence.

Outputs:

* data/output/log.txt → Summary of the last run
* data/output/most-feq.txt → Sorted list of words and their occurrence in the training corpus
* data/output/pos-neg-cnt.csv → List of words common between positive and negative sentence and number of their occurrence.

---------- Brief Summary ----------

**Main.py**

This is the entry to the code. All the set-up variables are assigned a default value according to the best results achieved in the tests. However, the default value can be changed by passing it as an argument to the main.py function.

For more information , please refer to the help provided in main.py, “Argument” section.

**data_prepare.py**

This is where the train and test data, are fetched, cleaned, tokenized and vectorized to be later used to train a support vector machine and a neural network.

**data_tokenizer.py**

This is where most of the text processing happens. Sentence elements (words in the default setting) are extracted. Chosen preprocessing are applied. A dictionary of all the information (e.g. word count, most frequent elements in the positive and negative sentences) is created. The information is finally used for text vectorization.

**model.py**

Two machine learning model are included in this code. One SVM and one NN. model.py includes the classes and all required modules to train, test and evaluate each ML model.

---------- ML choice justification ----------

**Why not rule-based?**

Considering the fact that there is no easy way to cover all possible word combinations as well as the availability of more than 500 tagged data entry for each class, rule-based models are not the best choices.

**Why SVM?**

SVMs are proven to be successful in binary classification. Their weakness is the need for feature engineering. However, sentiment analysis is rather straightforward and the list of top frequent words in each category can be used as SVM features. Furthermore, Considering the train-set size (2k), SVM is a reasonable choice.

Validation [accuracy 0.845 f1 0.843] Test [accuracy 0.743 f1 0.752]

**Why NN?**

Convolutional neural networks are the state of the art in many language processing tasks. Their need for very large number of labeled samples is the main problem when working with any NN.

However, following results show CNN work rather well in sentiment analysis even with 2k training samples.

Validation [loss: 0.094 acc: 0.875 f1: 0.872] Test result [loss: 0.174 acc: 0.789 f1: 0.797]