Assignment 2

Text Classification Benchmarks

Date: 07-03-2024

Laura Givskov Rahbek

Description

This folder contains assignment 2 for Language Analytics. The objective of the assignment is to train benchmark machine learning classifiers on structured text data, using `scikit-learn`, make and save understandable outputs and models, and save the results in clear ways. More specifically, a `TfidfVectorizer` will be used to vectorize and extract features from the *Fake or Real News* dataset (more in the Data section below). These features will be used in training two binary classification models to classify news articles as either 'REAL' or 'FAKE', namely the `LogisticRegression` classifier and the `MLPClassifier`. For both classifiers, gridsearch will be implemented to identify the paramter values that leads to the best performing model. `GridSearchCV` will be used to implement this, in addition to implementing a five-fold cross validation in training. For both classifiers it can be specified which metric the model should be tuned for. Three scripts were made for this assignment, one which preprocessed and vectorised the data and one script for each of the two classifiers used. The three are described below:

The `vectorizer.py` script does the following:

- Loads and splits the data into a test and train set.
- Defines and saves a `TfidfVectorizer` to the `models` folder, the following parameters are defined:
  - ngram_range = (1,2), allowing for both single words and up to two-word combinations in the features.
  - lowercase = True, making all characters lowercase.
  - max_features = 500, the top 500 features are used.
  - max_df = 0.95 and min_df = 0.05, the five percent most and least frequent are ignored.
- Fits and vectorises the training data, vectroizes the test data, then saves the extracted features to the `out` folder.

Both classifier scripts first, load the vectorised features saved in the `out` folder. And lastly, they evaluate the performance of the model on the test data and saves the evaluation metrics to the `out` folder. Besides the classification report, the parameters used in the estimator can also be viewed in this file. The `MLP_classifier.py` also saves a plot of the training loss and validation accuracy.

After loading in the data the `LR_classifier.py` script defines and fits a `LogisticRegression` classifier to the training data. Whether or not gridsearch is implemented, `max_iter` = 1000 and `random_state` = 42, and the final fitted model is saved to the `models` folder. The parameter used are described below:

- If gridsearch is not implemented all paremeters are kept at their default values, available at the documentation.
- If gridsearch is implemented, `solver`, `penalty`, `C` and `tol` are tuned, the remaining parameters are kept at their default values.
  - `solver` defines the optimization algorithm.'lbfgs', 'saga' and 'liblinear were included; 'lbfgs' is robust and the default solver, 'liblinear' is recommended on smaller datasets, and 'saga' is overall well performing.
  - `penalty` determines the regularization technique implemented, helping to balance between model fit and complexity. Different penalties are available for different solvers, leading to choosing 'l1', 'l2' and None.
  - `C` defines the strength of regularization; the larger the value, the less regulated the model is. The default is 1.0, additionally 0.1 and 0.01 are included.
  - `tol` defines the threshold for when the model should stop training, the default is 0.0001, 0.00001 and 0.001 are included as well.

After loading in the data the `MLP_classifier.py` script defines and fits a `MLPClassifier` to the training data. Whether or not gridsearch is implemented; `solver` = 'adam', `max_iter` = 1000, `random_state` = 42 and `early_stopping` = True (10% of the training data is used in validation, and the model stops training when the accuracy on the validation set does not increase by `tol` for 10 epochs), and the fitted model is saved to the `models` folder. The parameters are described below:

- If gridsearch is not implemented; the additional parameters are kept at their default values, available at the documentation.
- If gridsearch is implemented; `activation`, `hidden_layer_sizes` and `tol` are tuned, the remaining parameters are kept at their default values.
  - `activation` defines how the nodes are activated, 'relu' and 'logistic' are included.
  - `hidden_layer_sizes` determines the sizes of the hidden layers, the default is 100. 50 and 75 were also included as values in the gridsearch. As with other parameters, the balance is between getting a too complex model that overfits (which can happen with too large a hidden layer size) and a too simple model that is underfitting (which can happen with too small a hidden layer size).

  - `tol` defines the threshold for when the model should stop training, the default is 0.0001, 0.00001 and 0.001 are included as well.

In addition to the these three scripts, the script `shap_plots.py` is also accesible in the `src` folder. It implements the SHAP library, which I have used to visualize the importance of different features in the classification scheme for the `LogisticRegression` classifier. As the library is relatively new, it can not at the moment work with the `MLPClassifier`. When running the script two arguments should be passed:

- -i (*index*) The index of the article that should be plotted
- -l (*LRCmodel*) The model that should be used (the name of the file without the 'joblib' extension)

When running the script, a `force_plot` will be generated using the vectorised features and the chosen model for the text given as input. When the plot is saved the label (REAL or FAKE) will be printed in the terminal.

Data

The data used in this assignment is the *Fake or Real News* dataset, which can be downloaded here. The .csv file include 6335 articles, each represented in a row containing the title, the text, and a label indicating whether the article is fake or real.

Usage and Reproducing of Analysis

To reproduce the analysis:

- Download the `fake_or_real_news.csv` file from the source given above, and place it in the `in` folder.
- Run `setup.sh` in the terminal, it creates a virtual environment and installs packages and dependencies in to it:

  ```
  bash setup.sh
  ```

- Open the virtual enviornment by writing the following in the terminal;

  ```
  source ./env/bin/activate
  ```

To run either of the two scripts; `LR_classifier.py` or `MLP_classifier.py`, it should be specified whether or not to perform gridsearch with the flag -g (*gridsearch*) and it can be specified which metric the gridsearch should be tuned for with the flag -s (*score*), the default is accuracy.

- To run `LR_classifier.py` with gridsearch and tunning for f1 write:

  ```
  python src/LR_classifier.py -g "GS" -s "f1"
  ```

- Or to run `MLP_classifier.py` with no gridsearch write:

  ```
  python src/MLP_classifier.py
  ```

- To run `shap_plots.py`, remain in the virtual envrionment and make sure that a `LogisticRegression` classifier has been saved. Then it can be run with the two flags -i and -l. E.g. to save a force plot of the features from the third article in the test data, using the model fitted with gridsearch to accuracy, run:

  ```
  python src/shap_plots.py -i 3 -l LRC_accuracy_GS
  ```

- To exit the virtual environment write `deactivate` in the terminal.

(Note: when both `LR_classifier.py` and `MLP_classifier.py` are run, they check if a file containing the vectorised data exists in the `out` folder, if not they will run the `main()` function from `vectorizer.py` directly and save the data. Alernatively, the `vectorizer.py` script can be run by itself, with `python src/vectorizer.py` in the terminal.)

Discussion

Before evaluating the results of the gridsearch, the two base-classifiers are compared (complete classification reports can be found in the `out` folder). Both models had an accuracy, macro and weighted average of precision recall and f1 of 0.89. The `MLPClassifier` has slighlty better recall for the 'REAL' texts and better precision for the 'FAKE' texts. The `MLPClassifier` takes longer and uses more resources than the `LogisticRegression` classifier. With these parameters and results, it is not justifiable to use the neural network instead of the logistic regression classifier (for more on the impact and resource use of the models from this assignment see `Assignment 5`).

The performance of the best performing model in the gridsearch for the `LogisticRegression` classifier, is slighlty worse than the base-model at 0.88. First of all, this is likely due to aggregating when evaluating the cross validation. However, even if the accuracy, macro and weighted averages of precision, recall and f1, were slighlty higher than 0.88, the minimal difference in performance cannot justify the implementation of grid-search. In this case, the default parameters performed very well on their own. Some evaluation metrics and the parameter values for the two models can be seen in the table below:

Logisitic Regression Classifier: Parameter Values and Evaluation Metrics

model	C	penalty	solver	tol	accuracy	precision	recall	f1
LRC_GS	1.0	l1	saga	0.00001	0.88	0.88	0.88	0.88
LRC_%GS	1.0	l2	lbfgs	0.0001	0.89	0.89	0.89	0.89

The performance of the best performing model in the gridsearch for the `MLPClassifier`, were even closer to the base-model performance, at 0.89. As stated, the cross-validation affects the results, as each parameter-combination was fitted five times. This can be seen clearly in the loss and validation accuracy plots for the base model and the cross-validated model. Implementing gridsearch did not help argue that the CNN should be used in the first

place, the `LogisiticRegression` classifier remains the best option for this data.

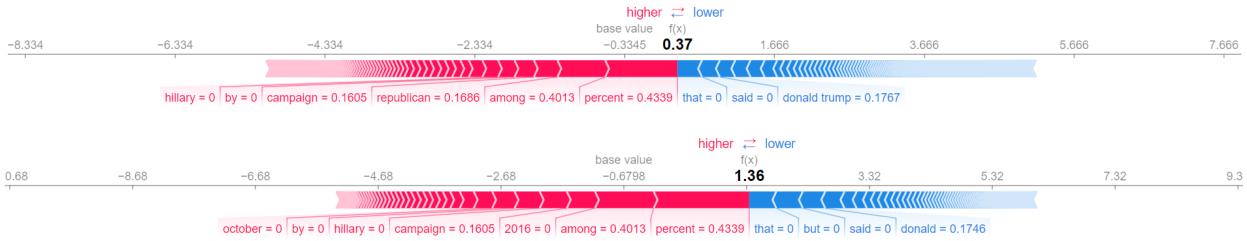
MLP Classifier: Parameter Values and Evaluation Metrics

model	activation	tol	hidden_layer_sizes	accuracy	precision	recall	f1
LRC_GS	relu	0.00001	75	0.89	0.89	0.89	0.89
LRC_%GS	relu	0.0001	100	0.89	0.89	0.89	0.89

It should be pointed out that the gridsearch is limited to the exact values given, which means that a better performance might have been found somewhere between some of these values. Tuning other parameters might also have introduced an increase in performance not gained with the chosen parameters in this assignment.

The `shap_plots.py` script was run on several samples from the test data, with both the `LogisiticRegression` classifier with and without gridsearch. All the resulting .html files is in `out/shap_plots/`. I have included two examples of the plots generated below. Both display the index 500 in the test data (label = REAL), the first plot is from the model without gridsearch and the second with gridsearch.

SHAP Force Plots



The bold number represent the prediction of the model with higher values leading the preditction to be 1 (REAL) and lower to be 0 (FAKE). The words below the red and blue bar are the TF-IDF Vectorizer features, and the values represent how much they contribute to the given score. E.g. In the top plot, 'donald trump' pulled the score downwards and 'percent' pulled it upwards. Interestingly, the absense of 'hillary' also pulled the score upwards. Both models correctly predicted the text in question as REAL, but the plots still look slighlty different, as different features in their respective trainings have been given weight. However overall, in this example they are very alike, which again reflects the classification reports conclusions that the models perform very similarly.

`codecarbon` was used to track the environmental impact when running this code, the results and an exploration of this can be found in the `Assignment-5` folder in the repository.