Sentiment-Aware News Classification

This framework provides the full pipeline for a sentiment-aware news classifier that reads preprocessed news data (headlines and articles) and returns an evaluation of whether the overall sentiment is positive, negative, or neutral.

Overview

In the following, the structure of the framework is described in detail and a manual for easy implementation is provided.

Table of Contents:

- Preparations
- The Dataset
- The Framework
- Tensorboard Logs
- Optimisation Documentation
- Output Models
- Preprocessing Data
- Prediction
- Run the Framework
- Results
- Extras
- GitHub Repository

Preparations

This section describes all necessary steps to create the correct working environment that ensures the functionality of this framework.

It is recommended to first create a virtual environment so that already existing systems or data on a device are not disturbed. The following steps need to be taken to establish the right environment for the classifier. It is expected that the latest version of Python 3.8 is already installed, together with pip.

1. Install Virtualenv:

```
$ pip3 install virtualenv virtualenvwrapper
```

2. Edit `~/.bashrc` profile

```
$ vim ~/.bashrc
```

and then:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.local/bin/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
export
VIRTUALENVWRAPPER_VIRTUALENV=$HOME/Library/Python/3.8/bin/virtualenv
source $HOME/Library/Python/3.8/bin/virtualenvwrapper.sh
```

3. Source venv

```
$ source ~/.bashrc
```

New terminal commands:

- Create an environment with `mkvirtualenv`
- Activate an environment (or switch to a different one) with `workon`
- Deactivate an environment with `deactivate`
- Remove an environment with `rmvirtualenv`

More information can be found in the docs.

4. Create new venv:

```
$ mkvirtualenv classifier -p python3
```

5. Install packages:

```
$ workon classifier
$ pip install keras-tuner
$ pip install numpy
$ pip install scikit-learn
$ pip install pandas
$ pip install numpy
$ pip install gensim
$ pip install plot_keras_history
$ pip install collections
$ pip install nltk
$ pip install seaborn
$ pip install spacy
$ pip install glob
$ pip install matplotlib
$ pip install joblib
$ pip install tensorflow # or tensorflow-gpu
```

Note:

A virtual environment is not necessary to run the framework, however, it is recommended to protect the data and devices.

If working without a virtual environment, please make sure that all necessary libraries are installed. If not, install them as follows:

```
$ pip3 install keras-tuner
$ pip3 install numpy
$ pip3 install scikit-learn
$ pip3 install pandas
$ pip3 install numpy
$ pip3 install gensim
$ pip3 install plot_keras_history
$ pip3 install collections
$ pip3 install nltk
$ pip3 install seaborn
$ pip3 install spacy
$ pip3 install glob
$ pip3 install matplotlib
$ pip3 install joblib
$ pip3 install tensorflow # or tensorflow-gpu
```

All these packages must be installed before running the classifier so that no errors occur.

The Dataset

The directory `info` contains the partial dataset that is used to train and test the classifier. The important files are:

- `titles_with_labels.csv`: This stores all news headlines with the four labels plus the aggregated label. The preprocessed text is also included, which is used for training purposes. Unfortunately, the article dataset is too big to be uploaded here.
- `abcnews_date_text.csv`: This stores the Australian news headlines used for predictions.

The structure of the dataset is simple to support further extensions. The file `labels.csv` includes the information about the three possible sentiment classes:

- 0: Negative
- 1: Positive
- -1: Neutral

Each news headline is only labelled as one of these sentiments.

The Framework

The directory `framework` will offer the necessary functions and methods needed to build, train, test, and optimise the different models.

The framework can be used by specifying the intentions through the arguments in the Terminal. The files are created in an object-oriented structure to establish a clear infrastructure. The files work as follows:

- `arguments.py`: specify the arguments that are passed to `main.py`,
- `bayesian.py`: optimise the classifier with Bayesian Optimisation,
- `build_sk.py`: build the SK models using the scikit-learn libraries,
- `build_tf.py`: build the TF models using the TensorFlow libraries,
- `helper`: provide helping functions that can be used by each class,
- `hyperband.py`: optimise the classifier with Hyperband Optimisation,
- `inner_opt.py`: establish the inner optimisation method,
- `main.py`: run the framework as commanded through the Terminal,
- `predicting.py`: predict the labels of new images,
- `preprocessing.py`: preprocess the text and prepare them for the training, testing, or prediction processes if necessary,
- `randoms.py`: optimise the classifier with Random Search Optimisation,
- `read_data.py`: prepare the data to be used in further processes,
- `skip.py`: create a Skip-gram model to analyse word similarities,
- `training.py`: train the classifiers.

Tensorboard Logs

The folder `logs` entails all logs collected through the different training, testing, and optimisation stages of the classifiers. They can be used to analyse and evaluate the performance in TensorBoard. Further instructions can be found on Tensorboard's website.

Optimisation Documentation

During the automatic optimisation of the models, the three used methods (Bayesian Optimisation, Random Search, and Hyperband) documented their trials in the respective `_data` folders inside the `framework` directory.

Output Models

The `output` folder consists of the saved models. As there are several different models (with and without optimisation, and using various embeddings) it is necessary to separate them. With a final accuracy of 86%, the RNN model can be implemented immediately.

Preprocessing Data

This step is not necessary when using the provided datasets. However, if further preprocessing is wanted or a new dataset is introduced, then the preprocessing methods go through all the necessary steps to return clean tokens to be implemented in future processes.

Prediction

The predicted news headlines will be loaded into the folder `predictions` into separate files depending on the label they received to simplify the evaluation process.

Run the Framework

The first step is to change file paths for the datasets in all necessary files. In this case, please adjust the `read_data.py` file and the `predicting.py` file, so it properly depicts the correct paths.

To run the framework successfully, several steps need to be included.

1. Start Virtual Environment (if necessary)

```
$ source ~/.bashrc
$ workon classifier
```

2. Change directories

```
$ cd m_thesis/framework
```

3. Specify arguments:

8 arguments need to be added when running the code. The `class Args()` can be seen in `arguments.py`.

The main structure of a command is as follows:

```
$ python3 main.py -m -v -op -inop -d -pre -tr -pa
```

where all arguments are optional. The arguments are described below:

- `-m` or `--model` to specify which type of model to use; there are 10 choices,
- `-v` or `--vector` to specify which type of vectoriser to use when implementing an SK model,
- `-op` or `--optimiser` to add the optimisation methods to use, can be either `bayesian`, `random`, or `hyperband`,
- `-inop` or `--inner_optimiser` to add the inner optimisation method, can be either `adam`, `sgd`, or `rms`,
- `-d` or `--dataset` to add whether the model should be trained on `titles` or `articles`; the default is `titles`, as the articles are too complex and the actual dataset cannot be included in this repository,
- `-pre` or `--preprocessing` to specify whether the dataset needs to be preprocessed first; the default is no,
- `-tr` or `--train` to indicate whether the hyperparameters should be optimised during the training process or not, or to skip the training entirely to just predict labels, can be either `all`, `none`, or `pred`,
- `-pa` or `--path` to add the path to the output model.

4. Run command in the Terminal

The framework can be run through a command in the terminal. The `main.py` file will only run with the correct arguments.

The following command will train and test an RNN model with the assigned titles dataset without any extra optimisation and predict labels in the end:

```
$ python3 main.py -m rnn -inop adam -d titles -tr none -pa
../output/rnn.model
```

When optimising the neural network, add the `optimiser` argument as shown below and change `-tr` to `all`.

Note: The `main.py` file includes example commands for easy access.

5. Run optimisations

Bayesian Optimisation:

This command will train, test, and optimise a model by tuning all hyperparameters using Bayesian Optimisation and the inner optimiser Adam.

```
$ python3 main.py -m cnn -op bayesian -d titles -tr all -pa
../output/bayesian_cnn.model
```

Hyperband Optimisation:

This command will train, test, and optimise a model by tuning all hyperparameters using Hyperband Optimisation and the inner optimiser Adam.

```
$ python3 main.py -m rnn -op hyperband -d titles -tr all -pa
../output/hyperband_rnn.model
```

Random Search:

This command will train, test, and optimise a model by tuning all hyperparameters using Random Search and the inner optimiser Adam.

```
$ python3 main.py -m lstm -op random -d titles -tr all -pa
../output/random_lstm.model
```

Make sure to always provide the correct model path in combination with the correct optimiser and model to avoid any confusion.

6. Run predictions only

If there is already a fully trained and optimised model that is saved on a specific path, the training process can be skipped, by directly using the collision-aware contact classifier to predict labels of provided images.

As the classifier is the main product of the framework, this is the default setting. Hence, by writing

```
$ python3 main.py
```

into the Terminal, it will use the default model, in this case, the RNN model, which achieves 86.0% accuracy. The framework already provides a directory with sample images that can be used as a new dataset for predictions.

This will save the predicted news headline labels in the separate `predictions` folder.

However, if the RNN is desired to be trained first, it can be run as follows:

```
python3 main.py -tr none
```

Note:

The fully trained models can only be used in the environment they were first created. If loading a model that has been trained in a different environment, an error might occur.

7. Run SK models

When running a model using the scikit-learn libraries it is important to note that their infrastructure differs from the simple TF models, as a specific vectoriser needs to be specified in the arguments now. The below command trains a Logistic Regression model using a CountVectorizer.

```
$ python3 main.py -m log -v count -d titles -tr none -pa
../output/log_count.model
```

It is important to note that the hyperparameter optimisation methods do not apply to the SK models, thus the argument `-tr` always needs to be set to `none`. Additionally, the experiments showed that the Word2Vec models stopped performing on datasets larger than a few 1000 entries. It is recommended to use the `w2v` vectoriser only for a small dataset, which will then not lead to good accuracy. If the size of the dataset needs to be adjusted, please change it in the function `get_data()` in the file `read_data.py`. Moreover, as the predictions are mostly geared towards the TF models, it is recommended to apply the prediction process only to TF models.

Results

The output in the shell will indicate the current state of the programme.

A classification report will appear with the prediction accuracy if the training option is chosen.

All necessary data regarding performance and evaluation, as well as predictions, are documented in markdown reports in the `reports` directory. If further insights into the training and testing processes are wanted, all created plots are saved in the `plots` directory.

Extras

Skip-gram model

If using the framework to analyse the dataset and word similarities, the aforementioned command structure can be ignored. All that is needed to train a skip-gram model is the following command:

```
python3 main.py -m skip -pa ../output/skip.model
```

Jupyter Notebooks

The repository also includes a variety of Jupyter Notebooks that were run on Google Colab to access their GPUs due to the needed computing units to train the DistilBERT models. Thus, to run the DistilBERT models, the `bert.ipynb` file in the `framework` directory can be opened. If further insights into the labelling process are wanted, the file `label_comparison.ipynb` in the `info` directory can be executed.

GitHub Repository

This is the link to the entire GitHub repository.