Sentiment Classifier Documentation

Eric Daoud

Apr 19, 2019

Contents:

CHAPTER 1

About

The goal of this project was to create a sentiment classifier API that could use various models and datasets.

It is written in Python and uses the following libraries:

- Flask: for the API
- Tensorflow & Keras: for Machine Learning

For more details about the project, you can refer to these slides.

So far we are only using the IMDB large movie review dataset. But we plan to use more datasets later on.

CHAPTER 2

Installation

Here are the required steps to get started with the API:

- Clone the repository
- Download the IMDB dataset and place it in the data folder. We use pre-trained word embeddings from FastText, so you might want to download them to the data folder as well:
  - Link to the IMDB Large Movie Review dataset
  - Link to the FastText embeddings
- Create a virtual environment, and install the requirements from `requirements.txt` file
- Add "sentiment_classifier" to your `PYTHONPATH`:

```
export PYTHONPATH=.:$PYTHONPATH
```

- Train the models by running:

```
python sentiment_classifier/scripts/train.py
```

- Run the API:

```
python sentiment_classifier/api/wsgi.py
```

- Test the API:

```
import requests

r = requests.post(
  "http://localhost:8000/api/classify",
  json={"text": "I love it"}
)
```

CHAPTER 3

Getting Started

Make sure to checkout this notebook to better understand how the code works: Example Model Notebook.

To train the classifiers, run the `train.py` scripts located in `sentiment_classifier/scripts`.

You can also refer to the documentation.

3.1 sentiment_classifier

3.1.1 sentiment_classifier package

Subpackages

sentiment_classifier.api package

We use Flask to write the API and we are using the factory pattern to create the Flask application. This is an elegant method that allows us to separate the code for the app creation, and register all the blueprints in one place.

The factory runs the following steps:

- Create the Flask object
- Load the ML models and attach them
- Register the index blueprint

`sentiment_classifier.api.`**`create_app`**(*model_filepath*)
Flask app factory method

Returns The created Flask application

Submodules

sentiment_classifier.api.index module

Index blueprint for the Flask API. This blueprint hosts the code for classifying a sequence.

sentiment_classifier.api.index.**classify**()
api method for predicting the sentiment on a given sequence.

Returns A json with text, sentiment and score

sentiment_classifier.api.index.**index**()
api status check method.

Returns json

sentiment_classifier.api.wsgi module

sentiment_classifier.nlp package

Subpackages

sentiment_classifier.nlp.models package

This is the package where we keep the Machine Learning models.

So far we have different modules in it:

- model: module where the base class Model is defined. Every new Machine Learning model should inherit from it. It is an abstract class that provides the basic methods for training and making predictions.
- shallow_networks: module where we keep the shallow networks models, such as the basic LogisticRegression, or one hidden layer neural network.
- deep_networks: module for the deeper neural networks, like Recurrent Neural Nets or Convolutionnal ones.

Submodules

sentiment_classifier.nlp.models.deep_networks module

Code for deep neural networks models.

class sentiment_classifier.nlp.models.deep_networks.**BiLSTM**
Bases: *sentiment_classifier.nlp.models.model.Model*

build_model(*input_shape*)
Method for building the model.

Parameters **input_shape** (*int*) – Size of the input

Returns a keras model, to be compiled and trained

Return type model (keras.Models)

train(*reader*, *filepath*)
Method for training the model. Must be implemented by the subclasses.

Parameters

- **reader** (nlp.reader.Reader) – a Reader instance that contains the data to train the model on.
- **filepath** (*str*) – path to where the model will be stored

Returns None

sentiment_classifier.nlp.models.model module

Module containing the root Model class that every new model must inherit from.

The Model class has the following attributes:

- model: the ML model, so far built using Keras
- tokenizer: responsible for mapping words into indices

The Model class implements the following methods:

- build_model: builds the model
- train: trains the model
- save: saves the model weights & tokenizer
- predict: predicts on sentences
- _make_training_data: a private method that creates the train/test matrices from a Reader object

class sentiment_classifier.nlp.models.model.**Model**

Bases: abc.ABC

build_model(*input_shape*)

Method for building the model.

Parameters **input_shape** (*int*) – Size of the input

Returns a keras model, to be compiled and trained

Return type model (keras.Models)

load(*filepath*)

Load the model weights and tokenizer

Parameters **filepath** (*str*) – Path where to load the model.

predict(*texts*, *preprocessing_function*)

Predict on a sentence

Parameters

- **texts** (*np.ndarray*) – the texts to predict on
- **preprocessing_function** – a preprocessing function, from nlp.preprocessing module.

Returns the cleaned texts

Return type cleaned_texts(list)

save(*filepath*)

Save the model weights and tokenizer

Parameters **filepath** (*str*) – Path where to store the model.

train(*reader*, *filepath*)
Method for training the model. Must be implemented by the subclasses.

Parameters

- **reader** (nlp.reader.Reader) – a Reader instance that contains the data to train the model on.
- **filepath** (*str*) – path to where the model will be stored

Returns None

sentiment_classifier.nlp.models.shallow_networks module

Code for shallow neural networks models.

class sentiment_classifier.nlp.models.shallow_networks.**ExampleModel**
Bases: *sentiment_classifier.nlp.models.model.Model*

build_model(*input_shape*)
Method for building the model.

Parameters **input_shape** (*int*) – Size of the input

Returns a keras model, to be compiled and trained

Return type model (keras.Models)

train(*reader*, *filepath*)
Method for training the model. Must be implemented by the subclasses.

Parameters

- **reader** (nlp.reader.Reader) – a Reader instance that contains the data to train the model on.
- **filepath** (*str*) – path to where the model will be stored

Returns None

Submodules

sentiment_classifier.nlp.preprocessing module

Module for text pre-processing

We provide a basic test preprocessing function, that does the following tasks:

- Removes HTML
- Surround punctuation and special characters by spaces

This function can be passed to a Reader instance when loading the dataset.

Note: we did not lowercase the sentence, or removed the special characters on purpose. We think this information can make a difference in classifying sentiments. We are also using Word Embeddings, and the embeddings are different on lowercase vs uppercase words.

sentiment_classifier.nlp.preprocessing.**clean_text**(*text*)
Function to clean a string. This function does the following:

- Remove HTML tags

- Surround punctuation and special characters by spaces
- Remove extra spaces

Parameters **text** (*str*) – text to clean

Returns the cleaned text

Return type text (str)

sentiment_classifier.nlp.reader module

We are using the IMDB Large Movie Reviews dataset from Stanford AI.

It provides 50,000 reviews on movies, splitted half-half in train/test and labelled as positive or negative.

We provide an abstract class Reader that we can subclass for each dataset.

We do this to standardise the dataset loading, and make it easy to use multiple datasets in the rest of the code with a common interface.

The IMDBReader class implements all the code needed to load the IMDB dataset.

class sentiment_classifier.nlp.reader.**IMDBReader**(*path*)
Bases: *sentiment_classifier.nlp.reader.Reader*

load_dataset(*limit=None*, *preprocessing_function=None*)
Load the IMDB dataset.

This function can also:

- preprocess using a custom function
- set a maximum number of files to load

Parameters

- **limit** (*int, optional*) – Defaults to None. Max number of files to load.
- **preprocessing_function** (*optional*) – Defaults to None. Function for preprocessing the texts. No preprocessing by default.

class sentiment_classifier.nlp.reader.**Reader**(*path*)
Bases: abc.ABC

load_dataset(*path*, *limit=None*, *preprocessing_function=None*)

sentiment_classifier.nlp.tokenizer module

This module abstracts the tokenizer object, so that we can use tokenizers from different libraries and provide the same interface. Hence, we won't need to change the rest of the code when changing tokenizers.

So far we only have one tokenizer, based on keras.preprocessing.text.Tokenizer.

class sentiment_classifier.nlp.tokenizer.**BaseTokenizer**
Bases: abc.ABC

fit(*train_data*)
Fit the tokenizer on the training data.

Parameters **train_data** (*list*) – List of texts to fit the tokenizer on.

load(*filepath*)
Load the tokenizer from disk

Parameters **filename** (*str*) – Path to load the tokenizer from

Returns the tokenizer itself, with loaded data

Return type self (*BaseTokenizer*)

save(*filename*)
Persist the tokenizer to disk

Parameters **filename** (*str*) – Path to save to.

transform(*data*)
Predict on data.

Parameters **data** (*list*) – List of texts to predict on

class sentiment_classifier.nlp.tokenizer.**KerasTokenizer**(*pad_max_len*, *lower=False*, *filters='tn'*)
Bases: *sentiment_classifier.nlp.tokenizer.BaseTokenizer*

fit(*train_data*)
Fit the tokenizer on the training data.

Parameters **train_data** (*list*) – List of texts to fit the tokenizer on.

transform(*data*)
Predict on data.

Parameters **data** (*list*) – List of texts to predict on

sentiment_classifier.nlp.utils module

sentiment_classifier.nlp.utils.**load_word_vectors**(*filepath*, *word_index*, *vector_size*)
Load word embeddings from a file.

Parameters

- **filepath** (*str*) – path to the embedding file
- **word_index** (*dict*) – word indices from the keras Tokenizer
- **vector_size** (*int*) – embedding dimension, must match the trained word vectors

Returns a matrix of size (len(word_index) * vector_size) that assigns each word to its learned embedding.

Return type embedding_matrix (np.ndarray)

sentiment_classifier.scripts package

Submodules

sentiment_classifier.scripts.train module

Script to train the classifiers.

Example usage:

```
python sentiment_classifier/scripts/train.py --models ExampleModel BiLSTM
```

sentiment_classifier.scripts.train.**main**()

Function in charge of training.

1. Parse arguments from the command line
2. Instanciate the requested models
3. Train them

sentiment_classifier.scripts.train.**parse_arguments**()

Parse arguments from command line.

Returns Parser object, with the arguments as attributes.

Return type argparse.ArgumentParser

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

T