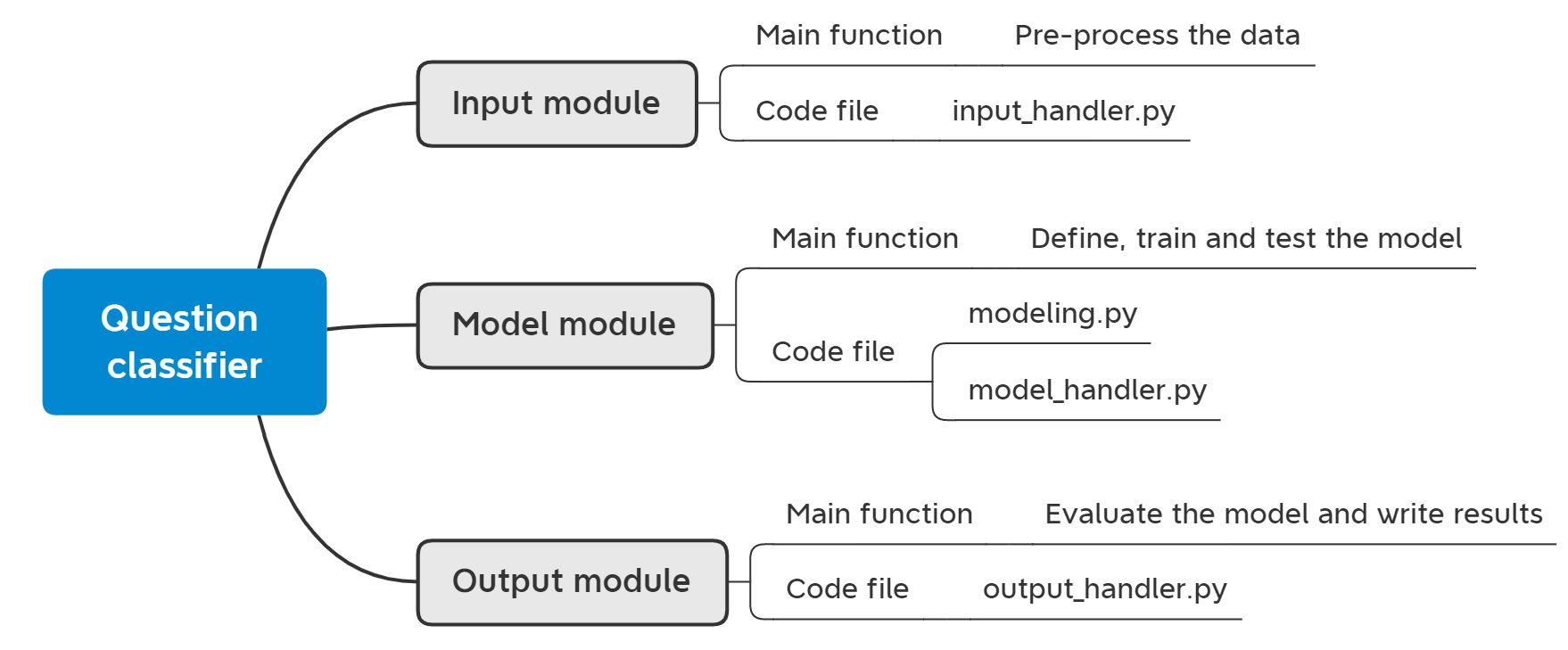
**Structure**



**Main code file:**

question\_classifier.py

**Dependency**

None

**Function:**

None

**Class:**

None

**Description:**

This module has no function and class. It just uses classes and functions implemented in other modules to build a standard machine learning workflow to deal with the question classification task:

1. Read configuration file.
2. Create a input handler to prepare the input for the model training and testing.
3. In training process: build, train and save the model; in testing process, load the model and get the predictions.
4. Create a output handler to do result evaluation and write results into the file.

**Input module:**

Input\_handler.py

**Dependency**

re

torch

random

pickle

numpy

Configparser

Sklearn(train\_test\_split from model\_selection)

**Function outside the Class:**

* print\_info(): print different configuration settings information at the beginning.
* load\_config\_file(config\_path): load the configuration file which stores all needed information.
* load\_stopwords(stopwords\_path): load the stopwords list used in pre-processing steps
* load\_mapping(mapping\_path): load char\_to\_id file or label\_to\_id file
  + Char\_to\_id/Label\_to\_id: dictionaries which map a word/label to a unique id
* load\_embedding\_dic(embedding\_path): load pre-trained embedding dictionary(glove)
* if\_contain\_non\_alphabet(word): check if a word contains non-alphabet characters
* remove\_unrelated\_words(text, stopwords): remove stopwords and words which contain non-alphabet characters
* pre\_process(raw\_text, stopwords): input a raw sentence and output a clean sentence by: (1)lowercase (2)remove punctuation (3)remove unrelated words
* load\_data(data\_path, stopwords): read the file and pre-process the data
* convert\_sentence\_to\_vectors(char\_to\_id, sentence, max\_length): input a text sentence and output a fixed-length(max\_length) tensor vectors by mapping each token to its unique id
* convert\_label\_to\_one\_hot\_encoding(label\_to\_id, label): input a text label and output a one-hot encoding tensor
* build\_data\_loader(sentence\_list, label\_list, batch\_size=1): build data loader for batch training

**Class:**

InputHandler

**Function inside the Class:**

* \_\_init\_\_(self, config): set up the intput handler such as storing the config, loading stopwords, char\_to\_id and label\_to\_id file.
* get\_training\_data(): load and pre-process the training/dev data
* get\_test\_data(): load and pre-process the test data
* get\_char\_to\_id(): return char\_to\_id dictionary
* transform\_input(sentences, labels): transform the data and prepare the input for model training and test
* get\_embedding\_matrix(): load pre-trained embedding file and create embedding matrix (used to initialize the embedding layer in neural network)

**Description:**

The main functions of this module are as follows:

1. Load and pre-process the raw data.
2. Load other files, such as char\_to\_id/label\_to\_id file, embedding file.
3. Transform and prepare the data that can be used in the model training and testing. For example, the data should be fixed-length tensors and label should be a one-hot encoding tensors

**Model module:**

modeling.py

**Dependency**

torch

random

**Class:**

BowClassifier

BiLSTMClassifier

BowBiLSTMClassifier

**Function of Class BowClassifier:**

* \_\_init\_\_(self, config, num\_words, embedding\_matrix=None): define the network structure of classifier based on BoW(bag of words) method
* forward(self, sentence): define the forward computation of the network based on BoW

**Function of Class BiLSTMClassifier:**

* \_\_init\_\_(self, config, num\_words, embedding\_matrix=None): define the network structure of classifier based on BiLSTMClassifier method
* forward(self, sentence): define the forward computation of the network based on BiLSTM

**Function of Class BowBiLSTMClassifier:**

* \_\_init\_\_(self, config, num\_words, embedding\_matrix=None): define the network structure of classifier based on BowBiLSTM method:
  + Combine the sentence vectors from BoW and BiLSTM through horizontal concatenation then send the new vector into the following classification layer
* forward(self, sentence): define the forward computation of the network based on BowBiLSTM

**Description:**

The main function of this code file in model module is as follows: define three different classifier models:

1. define their structure
2. define their forward computation steps

model\_handler.py

**Dependency**

os

torch

random

**Class:**

ModelHandler

**Function inside the class:**

* \_\_init\_\_(config, num\_words, embedding\_matrix, model\_path): set up the model handler such as storing the configuration object and build the model
* build\_model(num\_words, embedding\_matrix, model\_path): (1) build the model based on different selections (2) if model\_path is specified, then load the model.
* train(x\_train, y\_train, x\_dev, y\_dev): (1) build data loader (2) specify loss criteria and optimizer (3) start training: forward to get the prediction, backward to update the parameters (4) early stopping: calculate the accuracy on development set during each epoch, stop training if the best accuracy is not updated for a number of epochs (pre-defined)
* predict(x\_test): take a question as input and output the classification result through the forward computation of the model
* save\_model(): save the best model in training process
* load\_model(path): load pre-trained model in training process

**Description:**

The main function of this code file in model module is as follows: define the training, predicting, saving and loading steps of the model.

Especially for the training process:

1. Best accuracy: define a variable to store the best accuracy on dev set during training. Update it after each epoch
2. Early stopping: stop training if the best accuracy is not updated for a number of epochs (pre-defined)
3. Best model: save the best model (best accuracy on dev set) after the training.

**Output module:**

output\_handler.py

**Dependency**

pickle

numpy

matplotlib.pyplot

Sklearn.metrics (f1\_score, accuracy\_score, confusion\_matrix)

**Function outside the Class:**

* transform\_output(one\_hot\_predict, one\_hot\_true): transform the one-hot encoding tensor to numpy int label
* get\_id\_to\_label(path): create a id\_to\_label dictionary that maps a unique id to its label
* build\_classes(int\_predict, int\_true, id\_to\_label): get the text classes used for plotting confusion matrix and displaying f1 score of each class
* evaluate\_during\_training(one\_hot\_predict, one\_hot\_true): used for calculating the accuracy and f1 score during training
* plot\_confusion\_matrix\_figure(true, predict, classes, save\_img\_path): plot, display and save the confusion matrix

**Class:**

OutHandler

**Function inside the Class:**

* \_\_init\_\_(self, config, raw\_sentence): set up the output handler such as storing the config and initializing variables
* write\_result(): write the evaluation results into files
* result\_evaluation(one\_hot\_predict, one\_hot\_true): evaluate the model based on the prediction and true label

**Description:**

The main functions of this module are as follows:

1. Prepare for evaluation: Transform the output (the raw outputs are one-hot encoding tensors) from the model and prepare for evaluation
2. Result evaluation: calculate the accuracy, three types of F1 score (micro, macro, weighted), F1 score for each class, confusion matrix
3. Write the results: (1) accuracy and total three types of f1 score (2) f1 score of each class (3) write the original sentence, prediction label and true label for each testing question