ECE 601: Homework 10

Wei Xu
Email: xu1639@purdue.edu

Due date: 11:59 PM, Apr. 25, 2024
(Spring 2024)

1 Programming Tasks

1.1 Dataset

Note: Some of the code for this subsection is borrowed from the `DLStudio` package (https://engineering.purdue.edu/kak/distDLS/).

The code for loading data sets is shown below. The data sets are provided with `pkl` files, which are all dictionaries, and can be loaded using `pickle.load` method in the `load_dataset` function. There are 10,000 samples in total, and the proportion is 7:2:1 for training, evaluation, and testing sets respectively. Specifically, the `*_dict.pkl` files contain `'id'`, `'title'`, `'context'`, `'question'`, `'answers'`, while the `*_processed.pkl` files contain `'input_ids'`, `'attention_mask'`, `'start_positions'`, and `'end_positions'`.

```
# load datasets
def load_dataset(path='./dataset'):
    with open('{}/train_dict.pkl'.format(path), 'rb') as f:
        train_dict = pickle.load(f)
        f.close()
    with open('{}/eval_dict.pkl'.format(path), 'rb') as f:
        eval_dict = pickle.load(f)
        f.close()
    with open('{}/test_dict.pkl'.format(path), 'rb') as f:
        test_dict = pickle.load(f)
        f.close()
    with open('{}/train_data_processed.pkl'.format(path), 'rb') as f:
        train_processed = pickle.load(f)
        f.close()
    with open('{}/eval_data_processed.pkl'.format(path), 'rb') as f:
        eval_processed = pickle.load(f)
        f.close()
    with open('{}/test_data_processed.pkl'.format(path), 'rb') as f:
        test_processed = pickle.load(f)
        f.close()
    # print keys
    print(train_dict.keys())
    print(eval_dict.keys())
    print(test_dict.keys())
    print(train_processed.keys())
    print(eval_processed.keys())
    print(test_processed.keys())
    return train_dict, eval_dict, test_dict, train_processed,
        ↪ eval_processed, test_processed
```

1.2 BERT for Q&A

Note: Some of the code for this subsection is borrowed from the `DLStudio` package (https://engineering.purdue.edu/kak/distDLS/).

The code for fine-tuning a model is shown below. In the `train_model` function, the model (`'bert-base-uncased'`) is initialized through `BertForQuestionAnswering.from_pretrained`. The model structure can be inspected by printing `model._modules`. The training arguments are set through `TrainingArguments`. In this experiment, the number of epochs is 10, the batch size for training is 8, and the batch size for evaluation is 8. The log is printed every single epoch. The training and evaluation dataset loaders are created through `Dataset.from_pandas` with the `*_processed` sets. Then train the model using `Trainer`. And save the model after finishing training. The training output of the first 5 epochs can be found below the code snippet.

```
# fine tune model
def train_model(train_processed, eval_processed):
    # initialize model
    model_name = 'bert-base-uncased'
    model= BertForQuestionAnswering.from_pretrained(model_name)
    print(model._modules)
    # set training arguments
    training_args = TrainingArguments(output_dir='./results', # output
        ↪ directory
                                      use_mps_device=False,
                                      num_train_epochs=10, # total
                                         ↪ number of training epochs
                                      per_device_train_batch_size=8, #
                                         ↪ batch size per device
                                         ↪ during training
                                      per_device_eval_batch_size=8, #
                                         ↪ batch size for evaluation
                                      weight_decay=0.01, # strength of
                                         ↪ weight decay
                                      logging_strategy='epoch',
                                      logging_steps=1
                                      )
    # create dataset instance
    train_dataset = Dataset.from_pandas(pd.DataFrame(train_processed))
    eval_dataset = Dataset.from_pandas(pd.DataFrame(eval_processed))
    # train model
    trainer = Trainer(model=model, # the instantiated Transformers
        ↪ model to be fine-tuned
                      args=training_args, # training arguments, defined
                         ↪ above
                      train_dataset=train_dataset, # training dataset
                      eval_dataset=eval_dataset # evaluation dataset
                      )
    trainer.train()
    # save trainer instance
    trainer.save_model('./results/last_model')
```

Outputs:

```
{'loss': 2.0602, 'grad_norm': 34.05586242675781, 'learning_rate': 4.5e-05,
   ↪ 'epoch': 1.0}
```

```
{'loss': 0.9359, 'grad_norm': 36.63857650756836, 'learning_rate': 4e-05, '
    ↪ epoch': 2.0}
{'loss': 0.4943, 'grad_norm': 23.20585060119629, 'learning_rate': 3.5e-05,
    ↪  'epoch': 3.0}
{'loss': 0.2907, 'grad_norm': 44.92435836791992, 'learning_rate': 3e-05, '
    ↪ epoch': 4.0}
{'loss': 0.2036, 'grad_norm': 46.06318664550781, 'learning_rate': 2.5e-05,
    ↪  'epoch': 5.0}
...
{'train_runtime': 1687.3029, 'train_samples_per_second': 41.486, '
    ↪ train_steps_per_second': 5.186, 'train_loss': 0.43363596627371653, '
    ↪ epoch': 10.0}
```

The training loss keeps decreasing and the learning rate is decaying for more precise results.

1.3 Testing and evaluation metrics

Note: Some of the code for this subsection is borrowed from the `DLStudio` package (https://engineering.purdue.edu/kak/distDLS/).

The code for testing the model and calculating metrics is shown below. Call `test_model` to execute this process. The testing set is loaded through `Dataset.from_pandas`. The fine-tuned model is loaded through `BertForQuestionAnswering.from_pretrained` from the saved model. The tokenizer is created through `BertTokenizer.from_pretrained`. The prediction can be obtained by `trainer.predict`.

It is noticed that the predictions are all lowercase and in the subword-level tokenization. The preprocessing is needed. First, convert the prediction from subword-level tokenization to word-level tokenization by `subword2word`. This function simply removes `##`. Secondly, normalize the ground truth by `normalize_string`. This function converts all letters to lowercase ASCII characters and splits the words and symbols to match the prediction format. The degree symbol ° in `normalize_string` can not display well in the code snippet, so `^{\circ}` is used to represent it.

The Exact Match and F1-score metrics are calculated by `compute_exact_match` and `f1_score` respectively. The inputs are the prediction and ground truth after preprocessing.

15 samples and the (individual and overall) metrics can be found below the code snippet.

```python
# convert Unicode to ASCII
def unicode2ascii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# normalize sentence
def normalize_string(s):
    s = unicode2ascii(s.lower().strip())
    s = re.sub(r"([,.!?%$\-\'\'\"\"\(\)^{\circ}])", r" \1 ", s)
    s = re.sub(r"[^a-zA-Z0-9,.!?%$\-\'\'\"\"\(\)^{\circ}]+", r" ", s)
    s = re.sub(r'^\s+|\s+$', '', s)
    s = re.sub(r" ^{\circ} ", r"^{\circ}", s)
    return s

# convert subword-level token to word-level token
def subword2word(s):
    s = re.sub(r" ##", r"", s)
```

```
    return s

# compute Exact Match
def compute_exact_match(prediction, truth):
    return int(prediction == truth)

# compute F1-score
def f1_score(prediction, truth):
    pred_tokens = prediction.split()
    truth_tokens = truth.split()
    # if either the prediction or the truth is no-answer then F1 = 1 if
    ↪  they agree, 0 otherwise
    if len(pred_tokens) == 0 or len(truth_tokens) == 0:
        return int(pred_tokens == truth_tokens)
    else:
        common_tokens = set(pred_tokens) & set(truth_tokens)
        # if there are no common tokens then F1 = 0
        if len(common_tokens) == 0:
            return 0
        else:
            prec = len(common_tokens) / len(pred_tokens)
            rec = len(common_tokens) / len(truth_tokens)
            return 2 * (prec * rec) / (prec + rec)

# test model
def test_model(test_dict, test_processed):
    # create dataset instance
    test_dataset = Dataset.from_pandas(pd.DataFrame(test_processed))
    # load trained model
    model = BertForQuestionAnswering.from_pretrained('./results/
    ↪ last_model')
    trainer = Trainer(model=model)
    # Initialize the tokenizer
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    # test trained model
    x = trainer.predict(test_dataset)
    # retrieve offset mapping of prediction
    start_pos, end_pos = x.predictions
    start_pos = np.argmax(start_pos, axis=1)
    end_pos = np.argmax(end_pos, axis=1)
    # create lists to save metrics
    EM_list = []
    F1_list = []
    # print predictions and calculate metrics
    for k, (i, j) in enumerate(zip(start_pos, end_pos)):
        # convert indices to tokens
        tokens = tokenizer.convert_ids_to_tokens(test_processed['
        ↪ input_ids'][k])
        # preprocess prediction and ground truth
        prediction = subword2word(' '.join(tokens[i:j+1]))
        truth = normalize_string(test_dict['answers'][k]['text'][0])
        print('Question:', test_dict['question'][k])
        print('Answer:', prediction)
        print('Correct Answer:', truth)
```

```
        em = compute_exact_match(prediction, truth)
        f1 = f1_score(prediction, truth)
        print('Exact Match:', em)
        print('F1 Score:', f1)
        EM_list.append(em)
        F1_list.append(f1)
        print('---')
    # print overall metrics
    print('\n')
    print('Exact Match: average: {}; median: {}'.format(statistics.mean
        ↪ (EM_list), statistics.median(EM_list)))
    print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↪ F1_list), statistics.median(F1_list)))
```

Outputs:

```
...
Question: What type of imaging was used to study the relationship between
    ↪ humans and dogs?
Answer: mri
Correct Answer: magnetic resonance imaging
Normalized Correct Answer: magnetic resonance imaging
Exact Match: 0
F1 Score: 0
---
Question: How much money were possible changes to the Mexico City section
    ↪ of the film rumored to have saved the production?
Answer: $ 20 million
Correct Answer: $20 million
Normalized Correct Answer: $ 20 million
Exact Match: 1
F1 Score: 1.0
---
Question: Where did the Chinese government decide that parents who had
    ↪ lost children could go for free treatment?
Answer: fertility clinics
Correct Answer: fertility clinics
Normalized Correct Answer: fertility clinics
Exact Match: 1
F1 Score: 1.0
---
Question: What is the ultimate goal for Theravadins?
Answer: nibbana
Correct Answer: Nibb na
Normalized Correct Answer: nibbana
Exact Match: 1
F1 Score: 1.0
---
Question: What university was Lok-Ham Chan a professor at?
Answer: university of washington
Correct Answer: the University of Washington
Normalized Correct Answer: the university of washington
Exact Match: 0
F1 Score: 0.8571428571428571
---
```

```
Question: Who was the Mongol prince?
Answer: sakya pandita
Correct Answer: Godan
Normalized Correct Answer: godan
Exact Match: 0
F1 Score: 0
---
Question: How many people were buried in the collapsed schools?
Answer: 1 , 700
Correct Answer: 1,700
Normalized Correct Answer: 1 , 700
Exact Match: 1
F1 Score: 1.0
---
Question: Who mentored contestants in the fourteenth and fifteenth seasons
   ↪  of American Idol?
Answer: randy jackson
Correct Answer: Scott Borchetta
Normalized Correct Answer: scott borchetta
Exact Match: 0
F1 Score: 0
---
Question: Thomas Stritch was an editor of which publican from Notre Dame?
Answer: matthew fitzsimons , frederick crosson
Correct Answer: Review of Politics
Normalized Correct Answer: review of politics
Exact Match: 0
F1 Score: 0
---
Question: Paperback of the Year award from Bestsellers magazine was
   ↪ awarded when?
Answer: 1962
Correct Answer: 1962
Normalized Correct Answer: 1962
Exact Match: 1
F1 Score: 1.0
---
Question: Gray color is often called what when referring to dogs?
Answer: blue
Correct Answer: blue
Normalized Correct Answer: blue
Exact Match: 1
F1 Score: 1.0
---
Question: How many persons were still unaccounted for in Yingxiu?
Answer: around 3 , 000
Correct Answer: around 9,000
Normalized Correct Answer: around 9 , 000
Exact Match: 0
F1 Score: 0.75
---
Question: How were the semi-finalists divided in season four?
Answer: gender
Correct Answer: by gender
```

```
Normalized Correct Answer: by gender
Exact Match: 0
F1 Score: 0.6666666666666666
---
Question: Who was the host of American Idol in its fourteenth season?
Answer: ryan seacrest
Correct Answer: Ryan Seacrest
Normalized Correct Answer: ryan seacrest
Exact Match: 1
F1 Score: 1.0
---
Question: In what borough is Godiva based?
Answer: manhattan
Correct Answer: Manhattan
Normalized Correct Answer: manhattan
Exact Match: 1
F1 Score: 1.0
---
...
Exact Match: average: 0.572; median: 1.0
F1 Score: average: 0.7179283755599364; median: 1.0
```

Generally, the predictions are precise. There are three cases. (1) Some answers are accurate. For example, the prediction for the question ("How much money were possible changes to the Mexico City section of the film rumored to have saved the production?") is "$ 20 million", which is the same with the (normalized) correct answer; the prediction for the question ("Where did the Chinese government decide that parents who had lost children could go for free treatment?") is "fertility clinics", which is the same with the (normalized) correct answer. (2) Some other answers are good enough but have bad metrics. For example, the prediction for the question ("What type of imaging was used to study the relationship between humans and dogs?") is "mri", which the abbreviation of the (normalized) correct answer "magnetic resonance imaging" (so in my opinion, this prediction can also be considered as a correct answer); the prediction for the question ("What university was Lok-Ham Chan a professor at?") is "university of washington", which only has one less "the" than the (normalized) correct answer (this can also be considered as a correct answer). (3) Some answers are incorrect. For example, the prediction for the question ("Who was the Mongol prince?") is "sakya pandita", while the correct answer should be "godan"; the prediction for the question ("Who was the Mongol prince?") is "sakya pandita", while the correct answer should be "godan".

The average Exact Match is 0.572, meaning 57.2% predictions are exactly correct. The average F1-score is 0.718, which is a not-bad level. The medians for the metrics are both 1.0, meaning at least a half answers are perfect.

However, considering the case (2), in which the answers are good enough but have bad metrics, the real performance of the model should be better than what the metrics tell.

1.4 Comparison

Note: Some of the code for this subsection is borrowed from the `DLStudio` package (https://engineering.purdue.edu/kak/distDLS/).

The code for comparison with another fine-tuned model is shown below. The `compare_model` function loads `'distilbert-base-cased-distilled-squad'` model from Hugging Face. Then run it over the testing set and calculate the Exact Match and F1-score metrics.

15 samples and the (individual and overall) metrics can be found below the code snippet.

```
# compare with another open-source fine-tuned model
def compare_model(test_dict):
    # load model
    question_answerer = pipeline('question-answering', model='
        ↪ distilbert-base-cased-distilled-squad')
    # create lists to save metrics
    EM_list = []
    F1_list = []
    # print predictions and calculate metrics
    for i in range(len(test_dict['question'])):
        result = question_answerer(question=test_dict['question'][i],
            ↪ context=test_dict['context'][i])
        print('Question:', test_dict['question'][i])
        print('Answer:', result['answer'])
        print('Correct:', test_dict['answers'][i]['text'][0])
        em = compute_exact_match(result['answer'], test_dict['answers'
            ↪ ][i]['text'][0])
        f1 = f1_score(result['answer'], test_dict['answers'][i]['text'
            ↪ ][0])
        print('Exact Match:', em)
        print('F1 Score:', f1)
        EM_list.append(em)
        F1_list.append(f1)
        print('---')
    # print overall metrics
    print('\n')
    print('Exact Match: average: {}; median: {}'.format(statistics.mean
        ↪ (EM_list), statistics.median(EM_list)))
    print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↪ F1_list), statistics.median(F1_list)))
```

Outputs:

```
...
Question: What type of imaging was used to study the relationship between
    ↪ humans and dogs?
Answer: magnetic resonance imaging
Correct: magnetic resonance imaging
Exact Match: 1
F1 Score: 1.0
---
Question: How much money were possible changes to the Mexico City section
    ↪ of the film rumored to have saved the production?
Answer: $20 million
Correct: $20 million
Exact Match: 1
F1 Score: 1.0
---
Question: Where did the Chinese government decide that parents who had
    ↪ lost children could go for free treatment?
Answer: fertility clinics
Correct: fertility clinics
Exact Match: 1
F1 Score: 1.0
```

```
---
Question: What is the ultimate goal for Theravadins?
Answer: Nibb na
Correct: Nibb na
Exact Match: 1
F1 Score: 1.0
---
Question: What university was Lok-Ham Chan a professor at?
Answer: University of Washington
Correct: the University of Washington
Exact Match: 0
F1 Score: 0.8571428571428571
---
Question: Who was the Mongol prince?
Answer: Godan
Correct: Godan
Exact Match: 1
F1 Score: 1.0
---
Question: How many people were buried in the collapsed schools?
Answer: 1,700
Correct: 1,700
Exact Match: 1
F1 Score: 1.0
---
Question: Who mentored contestants in the fourteenth and fifteenth seasons
   ↪  of American Idol?
Answer: Scott Borchetta
Correct: Scott Borchetta
Exact Match: 1
F1 Score: 1.0
---
Question: Thomas Stritch was an editor of which publican from Notre Dame?
Answer: The Review of Politics
Correct: Review of Politics
Exact Match: 0
F1 Score: 0.8571428571428571
---
Question: Paperback of the Year award from Bestsellers magazine was
   ↪ awarded when?
Answer: 1962
Correct: 1962
Exact Match: 1
F1 Score: 1.0
---
Question: Gray color is often called what when referring to dogs?
Answer: blue
Correct: blue
Exact Match: 1
F1 Score: 1.0
---
Question: How many persons were still unaccounted for in Yingxiu?
Answer: 9,000
Correct: around 9,000
```

```
Exact Match: 0
F1 Score: 0.6666666666666666
---
Question: How were the semi-finalists divided in season four?
Answer: by gender
Correct: by gender
Exact Match: 1
F1 Score: 1.0
---
Question: Who was the host of American Idol in its fourteenth season?
Answer: Ryan Seacrest
Correct: Ryan Seacrest
Exact Match: 1
F1 Score: 1.0
---
Question: In what borough is Godiva based?
Answer: Manhattan
Correct: Manhattan
Exact Match: 1
F1 Score: 1.0
---
...
Exact Match: average: 0.769; median: 1.0
F1 Score: average: 0.8906788305864256; median: 1.0
```

The same three cases also occur, but the overall performance is better. It pays attention to the case of letters and does not need to normalize the ground truth before calculating the metrics. In terms of the metrics, the average of Exact Match is 0.769, and the average of F1-score is 0.891. They are both higher than the model fine-tuned by myself. And no surprise that the medians for the metrics are both 1.0.

2 Complete Source Code

The degree symbol ° in `normalize_string` can not display well in the code snippet, so {\circ} is used to represent it.

```python
import numpy as np
import random
import torch
import os
import argparse
import pickle
import pandas as pd
import unicodedata
import re
import statistics
from transformers import BertForQuestionAnswering, TrainingArguments,
    ↪ Trainer, BertTokenizer, pipeline
from datasets import Dataset

# set ramdom seed
def random_seed_setting(seed):
    random.seed(seed)
    torch.manual_seed(seed)
```

```
    torch.cuda.manual_seed(seed)
    np.random.seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmarks = False
    os.environ['PYTHONHASHSEED'] = str(seed)

# load datasets
def load_dataset(path='./dataset'):
    with open('{}/train_dict.pkl'.format(path), 'rb') as f:
        train_dict = pickle.load(f)
        f.close()
    with open('{}/eval_dict.pkl'.format(path), 'rb') as f:
        eval_dict = pickle.load(f)
        f.close()
    with open('{}/test_dict.pkl'.format(path), 'rb') as f:
        test_dict = pickle.load(f)
        f.close()
    with open('{}/train_data_processed.pkl'.format(path), 'rb') as f:
        train_processed = pickle.load(f)
        f.close()
    with open('{}/eval_data_processed.pkl'.format(path), 'rb') as f:
        eval_processed = pickle.load(f)
        f.close()
    with open('{}/test_data_processed.pkl'.format(path), 'rb') as f:
        test_processed = pickle.load(f)
        f.close()
    # print keys
    print(train_dict.keys())
    print(eval_dict.keys())
    print(test_dict.keys())
    print(train_processed.keys())
    print(eval_processed.keys())
    print(test_processed.keys())
    return train_dict, eval_dict, test_dict, train_processed,
        ↪ eval_processed, test_processed

# fine tune model
def train_model(train_processed, eval_processed):
    # initialize model
    model_name = 'bert-base-uncased'
    model= BertForQuestionAnswering.from_pretrained(model_name)
    print(model._modules)
    # set training arguments
    training_args = TrainingArguments(output_dir='./results', # output
        ↪ directory
                                      use_mps_device=False,
                                      num_train_epochs=10, # total
                                         ↪ number of training epochs
                                      per_device_train_batch_size=8, #
                                         ↪ batch size per device
                                         ↪ during training
                                      per_device_eval_batch_size=8, #
                                         ↪ batch size for evaluation
```

```
                                        weight_decay=0.01, # strength of
                                            ↪ weight decay
                                        logging_strategy='epoch',
                                        logging_steps=1
                                        )
    # create dataset instance
    train_dataset = Dataset.from_pandas(pd.DataFrame(train_processed))
    eval_dataset = Dataset.from_pandas(pd.DataFrame(eval_processed))
    # train model
    trainer = Trainer(model=model, # the instantiated Transformers
        ↪ model to be fine-tuned
                      args=training_args, # training arguments, defined
                          ↪  above
                      train_dataset=train_dataset, # training dataset
                      eval_dataset=eval_dataset # evaluation dataset
                      )
    trainer.train()
    # save trainer instance
    trainer.save_model('./results/last_model')

# convert Unicode to ASCII
def unicode2ascii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# normalize sentence
def normalize_string(s):
    s = unicode2ascii(s.lower().strip())
    s = re.sub(r"([,.!?%$\-\'\'\"\"\(\)^{\circ}])", r" \1 ", s)
    s = re.sub(r"[^a-zA-Z0-9,.!?%$\-\'\'\"\"\(\)^{\circ}]+", r" ", s)
    s = re.sub(r'^\s+|\s+$', '', s)
    s = re.sub(r" ^{\circ} ", r"^{\circ}", s)
    return s

# convert subword-level token to word-level token
def subword2word(s):
    s = re.sub(r" ##", r"", s)
    return s

# compute Exact Match
def compute_exact_match(prediction, truth):
    return int(prediction == truth)

# compute F1-score
def f1_score(prediction, truth):
    pred_tokens = prediction.split()
    truth_tokens = truth.split()
    # if either the prediction or the truth is no-answer then F1 = 1 if
        ↪  they agree, 0 otherwise
    if len(pred_tokens) == 0 or len(truth_tokens) == 0:
        return int(pred_tokens == truth_tokens)
    else:
```

```
        common_tokens = set(pred_tokens) & set(truth_tokens)
        # if there are no common tokens then F1 = 0
        if len(common_tokens) == 0:
            return 0
        else:
            prec = len(common_tokens) / len(pred_tokens)
            rec = len(common_tokens) / len(truth_tokens)
            return 2 * (prec * rec) / (prec + rec)

# test model
def test_model(test_dict, test_processed):
    # create dataset instance
    test_dataset = Dataset.from_pandas(pd.DataFrame(test_processed))
    # load trained model
    model = BertForQuestionAnswering.from_pretrained('./results/
        ↪ last_model')
    trainer = Trainer(model=model)
    # Initialize the tokenizer
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    # test trained model
    x = trainer.predict(test_dataset)
    # retrieve offset mapping of prediction
    start_pos, end_pos = x.predictions
    start_pos = np.argmax(start_pos, axis=1)
    end_pos = np.argmax(end_pos, axis=1)
    # create lists to save metrics
    EM_list = []
    F1_list = []
    # print predictions and calculate metrics
    for k, (i, j) in enumerate(zip(start_pos, end_pos)):
        # convert indices to tokens
        tokens = tokenizer.convert_ids_to_tokens(test_processed['
            ↪ input_ids'][k])
        # preprocess prediction and ground truth
        prediction = subword2word(' '.join(tokens[i:j+1]))
        truth = normalize_string(test_dict['answers'][k]['text'][0])
        print('Question:', test_dict['question'][k])
        print('Answer:', prediction)
        print('Correct Answer:', test_dict['answers'][k]['text'][0])
        print('Normalized Correct Answer:', truth)
        em = compute_exact_match(prediction, truth)
        f1 = f1_score(prediction, truth)
        print('Exact Match:', em)
        print('F1 Score:', f1)
        EM_list.append(em)
        F1_list.append(f1)
        print('---')
    # print overall metrics
    print('\n')
    print('Exact Match: average: {}; median: {}'.format(statistics.mean
        ↪ (EM_list), statistics.median(EM_list)))
    print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↪ F1_list), statistics.median(F1_list)))

```

```
# compare with another open-source fine-tuned model
def compare_model(test_dict):
    # load model
    question_answerer = pipeline('question-answering', model='
        ↪ distilbert-base-cased-distilled-squad')
    # create lists to save metrics
    EM_list = []
    F1_list = []
    # print predictions and calculate metrics
    for i in range(len(test_dict['question'])):
        result = question_answerer(question=test_dict['question'][i],
            ↪ context=test_dict['context'][i])
        print('Question:', test_dict['question'][i])
        print('Answer:', result['answer'])
        print('Correct:', test_dict['answers'][i]['text'][0])
        em = compute_exact_match(result['answer'], test_dict['answers'
            ↪ ][i]['text'][0])
        f1 = f1_score(result['answer'], test_dict['answers'][i]['text'
            ↪ ][0])
        print('Exact Match:', em)
        print('F1 Score:', f1)
        EM_list.append(em)
        F1_list.append(f1)
        print('---')
    # print overall metrics
    print('\n')
    print('Exact Match: average: {}; median: {}'.format(statistics.mean
        ↪ (EM_list), statistics.median(EM_list)))
    print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↪ F1_list), statistics.median(F1_list)))

if __name__ == '__main__':
    # '1' -- fine tune model
    # '2' -- test fine-tuned model
    # '3' -- compare with another fine-tuned model
    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--task', type=str, default='1',\
        help='choose a task', choices=['1','2','3'])
    args = parser.parse_args()

    # set random seed
    seed = 60146
    random_seed_setting(seed)

    train_dict, eval_dict, test_dict, train_processed, eval_processed,
        ↪ test_processed = load_dataset()
    if args.task == '1':
        train_model(train_processed, eval_processed)
    if args.task == '2':
        test_model(test_dict, test_processed)
    if args.task == '3':
        compare_model(test_dict)
```