

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'.

```
1 # load datasets
2 def load_dataset(path='./dataset'):
3     with open('{}/train_dict.pkl'.format(path), 'rb') as f:
4         train_dict = pickle.load(f)
5         f.close()
6     with open('{}/eval_dict.pkl'.format(path), 'rb') as f:
7         eval_dict = pickle.load(f)
8         f.close()
9     with open('{}/test_dict.pkl'.format(path), 'rb') as f:
10        test_dict = pickle.load(f)
11        f.close()
12    with open('{}/train_data_processed.pkl'.format(path), 'rb') as f:
13        train_processed = pickle.load(f)
14        f.close()
15    with open('{}/eval_data_processed.pkl'.format(path), 'rb') as f:
16        eval_processed = pickle.load(f)
17        f.close()
18    with open('{}/test_data_processed.pkl'.format(path), 'rb') as f:
19        test_processed = pickle.load(f)
20        f.close()
21    # print keys
22    print(train_dict.keys())
23    print(eval_dict.keys())
24    print(test_dict.keys())
25    print(train_processed.keys())
26    print(eval_processed.keys())
27    print(test_processed.keys())
28    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.

```
1 # fine tune model
2 def train_model(train_processed, eval_processed):
3     # initialize model
4     model_name = 'bert-base-uncased'
5     model= BertForQuestionAnswering.from_pretrained(model_name)
6     print(model._modules)
7     # set training arguments
8     training_args = TrainingArguments(output_dir='./results', # output
          ↪ directory
9
10                                     use_mps_device=False,
11                                     num_train_epochs=10, # total
12                                     ↪ number of training epochs
13                                     per_device_train_batch_size=8, #
14                                     ↪ batch size per device
15                                     ↪ during training
16                                     per_device_eval_batch_size=8, #
17                                     ↪ batch size for evaluation
18                                     weight_decay=0.01, # strength of
19                                     ↪ weight decay
20                                     logging_strategy='epoch',
21                                     logging_steps=1
22                                     )
23     # create dataset instance
24     train_dataset = Dataset.from_pandas(pd.DataFrame(train_processed))
25     eval_dataset = Dataset.from_pandas(pd.DataFrame(eval_processed))
26     # train model
27     trainer = Trainer(model=model, # the instantiated Transformers
28          ↪ model to be fine-tuned
29                       args=training_args, # training arguments, defined
30          ↪ above
31                       train_dataset=train_dataset, # training dataset
32                       eval_dataset=eval_dataset # evaluation dataset
33                       )
34     trainer.train()
35     # save trainer instance
36     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 $^{\circ}$ 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.

```
1 # convert Unicode to ASCII
2 def unicode2ascii(s):
3     return ''.join(
4         c for c in unicodedata.normalize('NFD', s)
5         if unicodedata.category(c) != 'Mn'
6     )
7
8 # normalize sentence
9 def normalize_string(s):
10    s = unicode2ascii(s.lower().strip())
11    s = re.sub(r"([.!?%$-\'\'\'\\""\(\)\^{\circ}])", r" \1 ", s)
12    s = re.sub(r"^[a-zA-Z0-9,.!%$-\'\'\'\\""\(\)\^{\circ}]+", r" ", s)
13    s = re.sub(r"^\s+|\s+$", '', s)
14    s = re.sub(r" ^{\circ} ", r"^{\circ}", s)
15    return s
16
17 # convert subword-level token to word-level token
18 def subword2word(s):
19    s = re.sub(r" ##", r"", s)
```

```

20     return s
21
22 # compute Exact Match
23 def compute_exact_match(prediction, truth):
24     return int(prediction == truth)
25
26 # compute F1-score
27 def f1_score(prediction, truth):
28     pred_tokens = prediction.split()
29     truth_tokens = truth.split()
30     # if either the prediction or the truth is no-answer then F1 = 1 if
    ↪ they agree, 0 otherwise
31     if len(pred_tokens) == 0 or len(truth_tokens) == 0:
32         return int(pred_tokens == truth_tokens)
33     else:
34         common_tokens = set(pred_tokens) & set(truth_tokens)
35         # if there are no common tokens then F1 = 0
36         if len(common_tokens) == 0:
37             return 0
38         else:
39             prec = len(common_tokens) / len(pred_tokens)
40             rec = len(common_tokens) / len(truth_tokens)
41             return 2 * (prec * rec) / (prec + rec)
42
43 # test model
44 def test_model(test_dict, test_processed):
45     # create dataset instance
46     test_dataset = Dataset.from_pandas(pd.DataFrame(test_processed))
47     # load trained model
48     model = BertForQuestionAnswering.from_pretrained('./results/
    ↪ last_model')
49     trainer = Trainer(model=model)
50     # Initialize the tokenizer
51     tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
52     # test trained model
53     x = trainer.predict(test_dataset)
54     # retrieve offset mapping of prediction
55     start_pos, end_pos = x.predictions
56     start_pos = np.argmax(start_pos, axis=1)
57     end_pos = np.argmax(end_pos, axis=1)
58     # create lists to save metrics
59     EM_list = []
60     F1_list = []
61     # print predictions and calculate metrics
62     for k, (i, j) in enumerate(zip(start_pos, end_pos)):
63         # convert indices to tokens
64         tokens = tokenizer.convert_ids_to_tokens(test_processed['
    ↪ input_ids'][k])
65         # preprocess prediction and ground truth
66         prediction = subword2word(' '.join(tokens[i:j+1]))
67         truth = normalize_string(test_dict['answers'][k]['text'][0])
68         print('Question:', test_dict['question'][k])
69         print('Answer:', prediction)
70         print('Correct Answer:', truth)

```

```

71     em = compute_exact_match(prediction, truth)
72     f1 = f1_score(prediction, truth)
73     print('Exact Match:', em)
74     print('F1 Score:', f1)
75     EM_list.append(em)
76     F1_list.append(f1)
77     print('---')
78     # print overall metrics
79     print('\n')
80     print('Exact Match: average: {}; median: {}'.format(statistics.mean(
      ↪ (EM_list), statistics.median(EM_list)))
81     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.

```
1 # compare with another open-source fine-tuned model
2 def compare_model(test_dict):
3     # load model
4     question_answerer = pipeline('question-answering', model='
    ↪ distilbert-base-cased-distilled-squad')
5     # create lists to save metrics
6     EM_list = []
7     F1_list = []
8     # print predictions and calculate metrics
9     for i in range(len(test_dict['question'])):
10         result = question_answerer(question=test_dict['question'][i],
    ↪ context=test_dict['context'][i])
11         print('Question:', test_dict['question'][i])
12         print('Answer:', result['answer'])
13         print('Correct:', test_dict['answers'][i]['text'][0])
14         em = compute_exact_match(result['answer'], test_dict['answers'
    ↪ ][i]['text'][0])
15         f1 = f1_score(result['answer'], test_dict['answers'][i]['text'
    ↪ ][0])
16         print('Exact Match:', em)
17         print('F1 Score:', f1)
18         EM_list.append(em)
19         F1_list.append(f1)
20         print('---')
21     # print overall metrics
22     print('\n')
23     print('Exact Match: average: {}; median: {}'.format(statistics.mean
    ↪ (EM_list), statistics.median(EM_list)))
24     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 $^\circ$ in `normalize_string` can not display well in the code snippet, so `{\circ}` is used to represent it.

```

1 | import numpy as np
2 | import random
3 | import torch
4 | import os
5 | import argparse
6 | import pickle
7 | import pandas as pd
8 | import unicodedata
9 | import re
10 | import statistics
11 | from transformers import BertForQuestionAnswering, TrainingArguments,
    |     ↪ Trainer, BertTokenizer, pipeline
12 | from datasets import Dataset
13 |
14 | # set random seed
15 | def random_seed_setting(seed):
16 |     random.seed(seed)
17 |     torch.manual_seed(seed)

```

```

18     torch.cuda.manual_seed(seed)
19     np.random.seed(seed)
20     torch.backends.cudnn.deterministic = True
21     torch.backends.cudnn.benchmarks = False
22     os.environ['PYTHONHASHSEED'] = str(seed)
23
24     # load datasets
25     def load_dataset(path='./dataset'):
26         with open('{} /train_dict.pkl'.format(path), 'rb') as f:
27             train_dict = pickle.load(f)
28             f.close()
29         with open('{} /eval_dict.pkl'.format(path), 'rb') as f:
30             eval_dict = pickle.load(f)
31             f.close()
32         with open('{} /test_dict.pkl'.format(path), 'rb') as f:
33             test_dict = pickle.load(f)
34             f.close()
35         with open('{} /train_data_processed.pkl'.format(path), 'rb') as f:
36             train_processed = pickle.load(f)
37             f.close()
38         with open('{} /eval_data_processed.pkl'.format(path), 'rb') as f:
39             eval_processed = pickle.load(f)
40             f.close()
41         with open('{} /test_data_processed.pkl'.format(path), 'rb') as f:
42             test_processed = pickle.load(f)
43             f.close()
44         # print keys
45         print(train_dict.keys())
46         print(eval_dict.keys())
47         print(test_dict.keys())
48         print(train_processed.keys())
49         print(eval_processed.keys())
50         print(test_processed.keys())
51         return train_dict, eval_dict, test_dict, train_processed,
           ↪ eval_processed, test_processed
52
53     # fine tune model
54     def train_model(train_processed, eval_processed):
55         # initialize model
56         model_name = 'bert-base-uncased'
57         model = BertForQuestionAnswering.from_pretrained(model_name)
58         print(model._modules)
59         # set training arguments
60         training_args = TrainingArguments(output_dir='./results', # output
           ↪ directory
61
62                                     use_mps_device=False,
63                                     num_train_epochs=10, # total
           ↪ number of training epochs
64                                     per_device_train_batch_size=8, #
           ↪ batch size per device
           ↪ during training
65                                     per_device_eval_batch_size=8, #
           ↪ batch size for evaluation

```

```

65         weight_decay=0.01, #strength of
        ↪ weight decay
66         logging_strategy='epoch',
67         logging_steps=1
68     )
69     # create dataset instance
70     train_dataset = Dataset.from_pandas(pd.DataFrame(train_processed))
71     eval_dataset = Dataset.from_pandas(pd.DataFrame(eval_processed))
72     # train model
73     trainer = Trainer(model=model, # the instantiated Transformers
        ↪ model to be fine-tuned
74                     args=training_args, # training arguments, defined
        ↪ above
75                     train_dataset=train_dataset, # training dataset
76                     eval_dataset=eval_dataset # evaluation dataset
77                 )
78     trainer.train()
79     # save trainer instance
80     trainer.save_model('./results/last_model')
81
82     # convert Unicode to ASCII
83     def unicode2ascii(s):
84         return ''.join(
85             c for c in unicodedata.normalize('NFD', s)
86             if unicodedata.category(c) != 'Mn'
87         )
88
89     # normalize sentence
90     def normalize_string(s):
91         s = unicode2ascii(s.lower().strip())
92         s = re.sub(r"([,!%$\-\'\"\\\"\\\(\)\^\{\circ}])", r" \1 ", s)
93         s = re.sub(r"[^a-zA-Z0-9,.!%$\-\'\"\\\"\\\(\)\^\{\circ}]+" , r" ", s)
94         s = re.sub(r'\s+|\s+$', '', s)
95         s = re.sub(r" ^{\circ} ", r"^{\circ}", s)
96         return s
97
98     # convert subword-level token to word-level token
99     def subword2word(s):
100         s = re.sub(r" ##", r"", s)
101         return s
102
103     # compute Exact Match
104     def compute_exact_match(prediction, truth):
105         return int(prediction == truth)
106
107     # compute F1-score
108     def f1_score(prediction, truth):
109         pred_tokens = prediction.split()
110         truth_tokens = truth.split()
111         # if either the prediction or the truth is no-answer then F1 = 1 if
        ↪ they agree, 0 otherwise
112         if len(pred_tokens) == 0 or len(truth_tokens) == 0:
113             return int(pred_tokens == truth_tokens)
114         else:

```

```

115     common_tokens = set(pred_tokens) & set(truth_tokens)
116     # if there are no common tokens then F1 = 0
117     if len(common_tokens) == 0:
118         return 0
119     else:
120         prec = len(common_tokens) / len(pred_tokens)
121         rec = len(common_tokens) / len(truth_tokens)
122         return 2 * (prec * rec) / (prec + rec)
123
124 # test model
125 def test_model(test_dict, test_processed):
126     # create dataset instance
127     test_dataset = Dataset.from_pandas(pd.DataFrame(test_processed))
128     # load trained model
129     model = BertForQuestionAnswering.from_pretrained('./results/
        ↪ last_model')
130     trainer = Trainer(model=model)
131     # Initialize the tokenizer
132     tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
133     # test trained model
134     x = trainer.predict(test_dataset)
135     # retrieve offset mapping of prediction
136     start_pos, end_pos = x.predictions
137     start_pos = np.argmax(start_pos, axis=1)
138     end_pos = np.argmax(end_pos, axis=1)
139     # create lists to save metrics
140     EM_list = []
141     F1_list = []
142     # print predictions and calculate metrics
143     for k, (i, j) in enumerate(zip(start_pos, end_pos)):
144         # convert indices to tokens
145         tokens = tokenizer.convert_ids_to_tokens(test_processed['
            ↪ input_ids'][k])
146         # preprocess prediction and ground truth
147         prediction = subword2word(' '.join(tokens[i:j+1]))
148         truth = normalize_string(test_dict['answers'][k]['text'][0])
149         print('Question:', test_dict['question'][k])
150         print('Answer:', prediction)
151         print('Correct Answer:', test_dict['answers'][k]['text'][0])
152         print('Normalized Correct Answer:', truth)
153         em = compute_exact_match(prediction, truth)
154         f1 = f1_score(prediction, truth)
155         print('Exact Match:', em)
156         print('F1 Score:', f1)
157         EM_list.append(em)
158         F1_list.append(f1)
159         print('---')
160     # print overall metrics
161     print('\n')
162     print('Exact Match: average: {}; median: {}'.format(statistics.mean(
        ↪ (EM_list), statistics.median(EM_list)))
163     print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↪ F1_list), statistics.median(F1_list)))
164

```

```

165 # compare with another open-source fine-tuned model
166 def compare_model(test_dict):
167     # load model
168     question_answerer = pipeline('question-answering', model='
        ↳ distilbert-base-cased-distilled-squad')
169     # create lists to save metrics
170     EM_list = []
171     F1_list = []
172     # print predictions and calculate metrics
173     for i in range(len(test_dict['question'])):
174         result = question_answerer(question=test_dict['question'][i],
        ↳ context=test_dict['context'][i])
175         print('Question:', test_dict['question'][i])
176         print('Answer:', result['answer'])
177         print('Correct:', test_dict['answers'][i]['text'][0])
178         em = compute_exact_match(result['answer'], test_dict['answers'
        ↳ ][i]['text'][0])
179         f1 = f1_score(result['answer'], test_dict['answers'][i]['text'
        ↳ ][0])
180         print('Exact Match:', em)
181         print('F1 Score:', f1)
182         EM_list.append(em)
183         F1_list.append(f1)
184         print('---')
185     # print overall metrics
186     print('\n')
187     print('Exact Match: average: {}; median: {}'.format(statistics.mean
        ↳ (EM_list), statistics.median(EM_list)))
188     print('F1 Score: average: {}; median: {}'.format(statistics.mean(
        ↳ F1_list), statistics.median(F1_list)))
189
190 if __name__ == '__main__':
191     # '1' -- fine tune model
192     # '2' -- test fine-tuned model
193     # '3' -- compare with another fine-tuned model
194     parser = argparse.ArgumentParser()
195     parser.add_argument('-t', '--task', type=str, default='1',\
196         help='choose a task', choices=['1', '2', '3'])
197     args = parser.parse_args()
198
199     # set random seed
200     seed = 60146
201     random_seed_setting(seed)
202
203     train_dict, eval_dict, test_dict, train_processed, eval_processed,
        ↳ test_processed = load_dataset()
204     if args.task == '1':
205         train_model(train_processed, eval_processed)
206     if args.task == '2':
207         test_model(test_dict, test_processed)
208     if args.task == '3':
209         compare_model(test_dict)

```