

# Project: Aspect-Based Sentiment Analysis

Quoc-Thai Nguyen, Duong-Thuan Nguyen, và Quang-Vinh Dinh  
PR-Team: Minh-Châu Phạm, Hoàng-Nguyên Vũ và Đăng-Nhã Nguyễn

Ngày 5 tháng 2 năm 2024

## Phần I. Giới thiệu

$S_i$ : The  $\overset{\text{sp}^1 \text{ negative}}{\underset{a^1}{\text{price}}}$  was  $\overset{\text{sp}^2 \text{ positive}}{\underset{a^2}{\text{cab}}}$   $\overset{o^1}{\text{too high}}$ , but the  $\overset{o^2}{\text{amazing}}$ .

Subtask	Input	Output
Aspect Term Extraction (ATE)	$S_i$	$a^1, a^2$
Aspect Term Sentiment Classification (ATSC)	$S_i + a^1, S_i + a^2$	$\text{sp}^1, \text{sp}^2$
Aspect Sentiment Pair Extraction (ASPE)	$S_i$	$(a^1, \text{sp}^1), (a^2, \text{sp}^2)$
Aspect Oriented Opinion Extraction (AOOE)	$S_i + a^1, S_i + a^2$	$o^1, o^2$
Aspect Opinion Pair Extraction (AOPE)	$S_i$	$(a^1, o^1), (a^2, o^2)$
Aspect Opinion Sentiment Triplet Extraction (AOSTE)	$S_i$	$(a^1, o^1, \text{sp}^1), (a^2, o^2, \text{sp}^2)$

Hình 1: Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis)

Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis) là bài toán có nhiều ứng dụng hiện nay trong việc phân tích các khía cạnh khác nhau của các bình luận, đánh giá về các sản phẩm, dịch vụ,... Ví dụ minh họa được minh họa như hình 1, với câu đầu vào: "The price was too high, but the cab was amazing.". Trong câu này, có hai khía cạnh được đánh giá là: "price" và "amazing". Với khía cạnh "price" được đánh giá là "too high" nên sẽ là "negative" và với khía cạnh "positive" được đánh giá là "amazing" nên sẽ là "positive".

Dựa vào việc xác định các giá trị đầu ra của mô hình, chúng ta có thể có một số bài toán nhỏ hơn như sau:

1. Aspect Term Extraction (ATE) hoặc Aspect-Based Term Extraction: trích xuất các khía cạnh được đánh giá trong bình luận
2. Aspect Term Sentiment Classification (ATSC): dựa vào câu đầu vào và các khía cạnh được đánh giá trong bình luận để dự đoán cảm xúc của bình luận
3. Aspect Sentiment Pair Extraction (ASPE): dựa vào câu đầu vào, trích xuất ra khía cạnh và dự đoán cảm xúc của các khía cạnh
4. Aspect Oriented Opinion Extraction (AOOE): dựa vào câu đầu vào và khía cạnh, dự đoán đoạn văn bản thể hiện cảm xúc của khía cạnh
5. Aspect Opinion Pair Extraction (AOPE): dựa vào câu đầu vào trích xuất thông tin về khía cạnh và đoạn văn bản thể hiện cảm xúc của khía cạnh
6. Aspect Opinion Sentiment Triplet Extraction (AOSTE): dựa vào câu đầu vào, trích xuất các thông tin về khía cạnh, đoạn văn bản thể hiện cảm xúc của khía cạnh và cảm xúc của khía cạnh trong bình luận

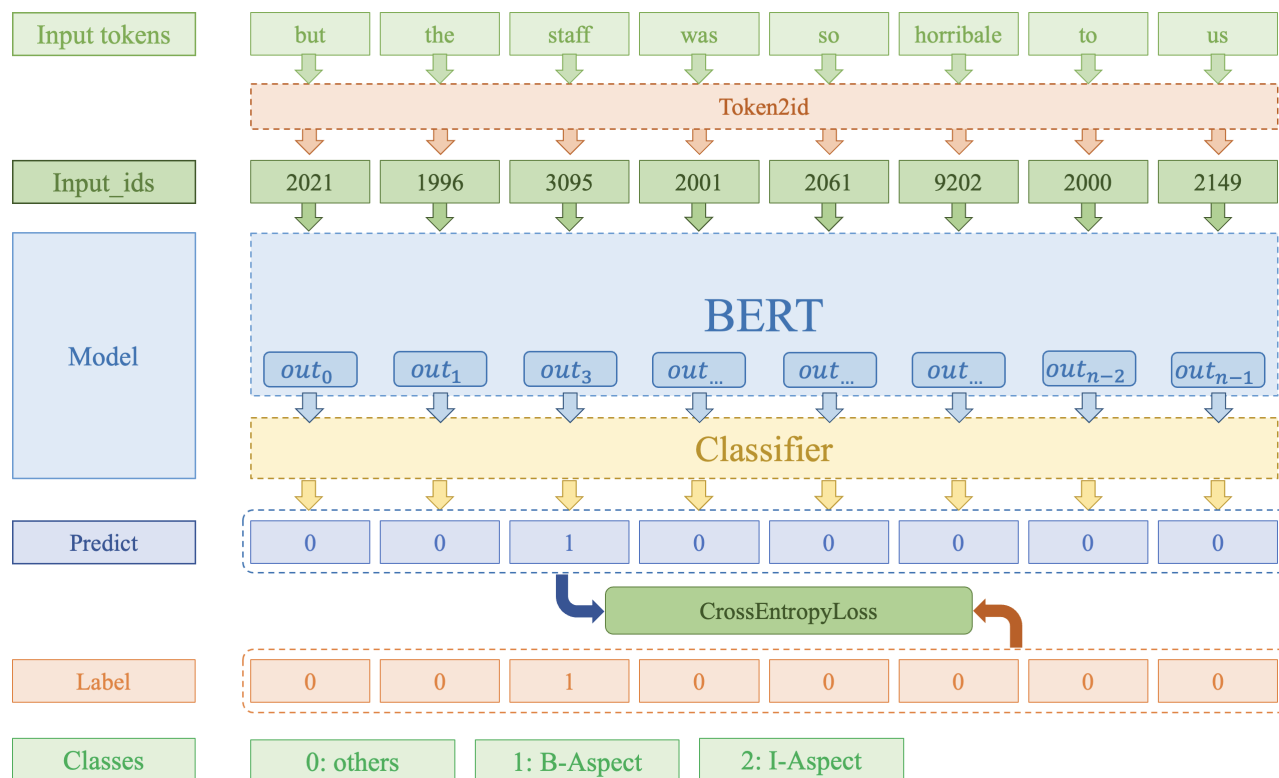
Trong phần này, chúng ta sẽ xây dựng mô hình giải quyết vấn đề cho bài toán ASPE. Dựa vào 2 bước chính:

1. Bước 1: Dự đoán từ trong văn bản thể hiện khía cạnh, hay chính là bài toán ATE
2. Bước 2: Dựa vào văn bản đầu vào và vị trí từ thể hiện khía cạnh dự đoán cảm xúc cho khía cạnh, hay là bài toán ATSC

Các thực nghiệm được xây dựng trên bộ dữ liệu **SemEval-2014 Task 4: Aspect Based Sentiment Analysis** và đã qua tiền xử lý cơ bản gồm: xóa bỏ dấu câu, chuẩn hoá và tách dựa vào khoảng trắng.

# Phần II. Aspect Term Extraction

Ở trong phần này, chúng ta xây dựng mô hình ATE dựa vào các phương pháp giải quyết bài toán sequence classification được sử dụng trong POS Tagging, NER,...



Hình 2: Aspect term extraction pipeline.

## 1. Build Dataset

```

1 # download dataset
2 !gdown 1d7JABk4jViI-USjLsWmhGkvzi8uQIL5C
3 !unzip ./data.zip
4
5 # load dataset
6 import pandas as pd
7
8 train_df = pd.read_csv('./data/restaurants_train.csv')
9 test_df = pd.read_csv('./data/restaurants_test.csv')
10
11 # tokenization
12 from transformers import BertTokenizer
13
14 model_name = "bert-base-uncased"
15 tokenizer = BertTokenizer.from_pretrained(model_name)
16
17 # build dataset
18 import torch
19 from torch.utils.data import Dataset
20
21 class ABSADataset(Dataset):

```

```

22     def __init__(self, df, tokenizer):
23         self.df = df
24         self.tokenizer = tokenizer
25
26     def __getitem__(self, idx):
27         tokens, tags, pols = self.df.iloc[idx, :3].values
28
29         tokens = tokens.replace("'", "").strip('"').split(',')
30         tags = tags.strip('['').split(',')
31         pols = pols.strip('['').split(',')
32
33         bert_tokens = []
34         bert_tags = []
35         bert_pols = []
36         for i in range(len(tokens)):
37             t = self.tokenizer.tokenize(tokens[i])
38             bert_tokens += t
39             bert_tags += [int(tags[i])] * len(t)
40             bert_pols += [int(pols[i])] * len(t)
41
42         bert_ids = self.tokenizer.convert_tokens_to_ids(bert_tokens)
43
44         ids_tensor = torch.tensor(bert_ids)
45         tags_tensor = torch.tensor(bert_tags)
46         pols_tensor = torch.tensor(bert_pols)
47         return bert_tokens, ids_tensor, tags_tensor, pols_tensor
48
49     def __len__(self):
50         return len(self.df)
51
52 train_ds = ABSADataset(train_df, tokenizer)
53 test_ds = ABSADataset(test_df, tokenizer)
54
55 # dataloader
56 from torch.nn.utils.rnn import pad_sequence
57
58 def padding(samples):
59     ids_tensors = [s[1] for s in samples]
60     ids_tensors = pad_sequence(ids_tensors, batch_first=True)
61
62     tags_tensors = [s[2] for s in samples]
63     tags_tensors = pad_sequence(tags_tensors, batch_first=True)
64
65     pols_tensors = [s[3] for s in samples]
66     pols_tensors = pad_sequence(pols_tensors, batch_first=True)
67
68     masks_tensors = torch.zeros(ids_tensors.shape, dtype=torch.long)
69     masks_tensors = masks_tensors.masked_fill(ids_tensors != 0, 1)
70
71     return ids_tensors, tags_tensors, pols_tensors, masks_tensors
72
73 from torch.utils.data import DataLoader
74
75 batch_size = 32
76 train_loader = DataLoader(
77     train_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
78 )
79 test_loader = DataLoader(
80     test_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
81 )

```

## 2. Modeling

```

1 from transformers import BertModel
2
3 class ABTEBert(torch.nn.Module):
4     def __init__(self, model_name):
5         super(ABTEBert, self).__init__()
6         self.bert = BertModel.from_pretrained(model_name)
7         self.linear = torch.nn.Linear(self.bert.config.hidden_size, 3)
8         self.loss_fn = torch.nn.CrossEntropyLoss()
9
10    def forward(self, ids_tensors, masks_tensors, tags_tensors):
11        bert_outputs= self.bert(
12            input_ids=ids_tensors, attention_mask=masks_tensors, return_dict=False
13        )
14        bert_outputs = bert_outputs[0]
15
16        linear_outputs = self.linear(bert_outputs)
17        if tags_tensors is not None:
18            tags_tensors = tags_tensors.view(-1)
19            linear_outputs_ = linear_outputs.view(-1,3)
20            loss = self.loss_fn(linear_outputs_, tags_tensors)
21            return loss, linear_outputs
22        else:
23            return linear_outputs

```

## 3. Trainer

```

1 import time
2 import numpy as np
3 from sklearn.metrics import classification_report
4
5 def train_epoch(model, optimizer, train_loader, device):
6     losses = []
7     for batch in (train_loader):
8         ids_tensors, tags_tensors, _, masks_tensors = batch
9         ids_tensors = ids_tensors.to(device)
10        tags_tensors = tags_tensors.to(device)
11        masks_tensors = masks_tensors.to(device)
12
13        loss, _ = model(
14            ids_tensors=ids_tensors,
15            masks_tensors=masks_tensors,
16            tags_tensors=tags_tensors
17        )
18        losses.append(loss.item())
19        loss.backward()
20        optimizer.step()
21        optimizer.zero_grad()
22    return sum(losses)/len(losses)
23
24 def evaluate_epoch(model, valid_loader, device):
25     losses = []
26
27     preds, labels = [], []
28     with torch.no_grad():
29         for batch in (valid_loader):
30             ids_tensors, tags_tensors, _, masks_tensors = batch
31             ids_tensors = ids_tensors.to(device)

```

```

32         tags_tensors = tags_tensors.to(device)
33         masks_tensors = masks_tensors.to(device)
34
35         loss, outputs = model(
36             ids_tensors=ids_tensors,
37             masks_tensors=masks_tensors,
38             tags_tensors=tags_tensors
39         )
40         losses.append(loss.item())
41
42         _, p = torch.max(outputs, dim=2)
43         preds += list([int(j) for i in p for j in i ])
44         labels += list([int(j) for i in tags_tensors for j in i ])
45
46     acc = np.mean(np.array(preds) == np.array(labels))
47     return sum(losses)/len(losses), acc
48
49 def train(model, model_name, save_model, optimizer, train_loader, valid_loader,
50 num_epochs, device):
51     train_losses = []
52     eval_accs, eval_losses = [], []
53     best_loss_eval = 100
54     times = []
55     for epoch in range(1, num_epochs+1):
56         epoch_start_time = time.time()
57         # Training
58         train_loss = train_epoch(model, optimizer, train_loader, device)
59         train_losses.append(train_loss)
60
61         # Evaluation
62         eval_loss, eval_acc = evaluate_epoch(model, valid_loader, device)
63         eval_accs.append(eval_acc)
64         eval_losses.append(eval_loss)
65
66         # Save best model
67         if eval_loss < best_loss_eval:
68             torch.save(model.state_dict(), save_model + f'/{model_name}.pt')
69
70         times.append(time.time() - epoch_start_time)
71         # Print loss, acc end epoch
72         print("-" * 59)
73         print(
74             "| End of epoch {:3d} | Time: {:.5.2f}s | Train Loss {:.8.3f} "
75             "| Valid Accuracy {:.8.3f} | Valid Loss {:.8.3f} ".format(
76                 epoch, time.time() - epoch_start_time, train_loss, eval_acc, eval_loss
77             )
78         )
79         print("-" * 59)
80
81     # Load best model
82     model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
83     model.eval()
84     metrics = {
85         'train_loss': train_losses,
86         'valid_accuracy': eval_accs,
87         'valid_loss': eval_losses,
88         'time': times
89     }
90     return model, metrics

```

#### 4. Training

```

1 save_model = "./model"
2 model = ABTEBert(model_name)
3 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
4 model.to(device)
5 optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
6 num_epochs = 5
7 best_model, metrics = train(
8     model, model_name, save_model, optimizer, train_loader, test_loader, num_epochs,
9     device
10 )

```

Kết quả training và accuracy trên tập test là 92.1

End of epoch	1	Time: 41.50s	Train Loss	0.284	Valid Accuracy	0.908	Valid Loss	0.233
End of epoch	2	Time: 36.46s	Train Loss	0.161	Valid Accuracy	0.916	Valid Loss	0.212
End of epoch	3	Time: 37.13s	Train Loss	0.100	Valid Accuracy	0.919	Valid Loss	0.249
End of epoch	4	Time: 37.27s	Train Loss	0.049	Valid Accuracy	0.914	Valid Loss	0.305
End of epoch	5	Time: 37.50s	Train Loss	0.020	Valid Accuracy	0.921	Valid Loss	0.338

Hình 3: Quá trình huấn luyện mô hình ATE.

## 5. Prediction

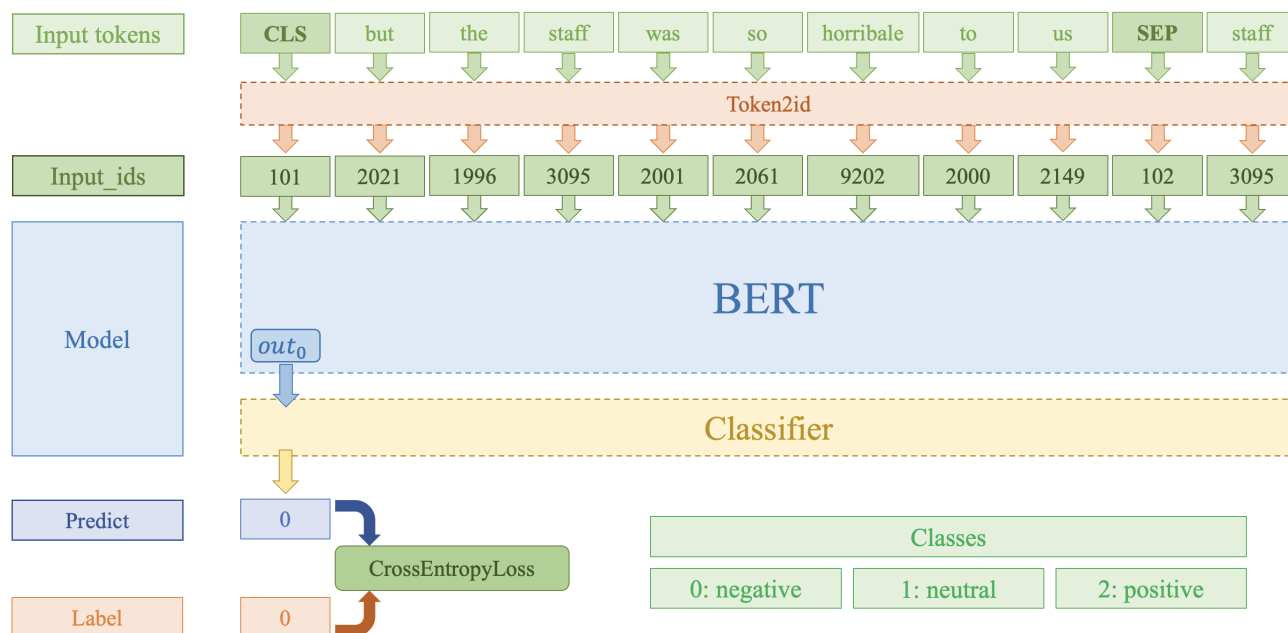
Sau quá trình huấn luyện và mô hình có độ đo đánh giá tốt nhất được sử dụng để gán nhãn cho các câu đầu vào

```
1 def predict(best_model, sentence, device):
2     word_pieces = list(tokenizer.tokenize(sentence))
3     input_ids = tokenizer.convert_tokens_to_ids(word_pieces)
4     input_tensor = torch.tensor([input_ids]).to(device)
5
6     with torch.no_grad():
7         outputs = model(input_tensor, None, None)
8         _, predictions = torch.max(outputs, dim=2)
9
10    predictions = predictions[0].tolist()
11    return word_pieces, predictions, outputs
12
13 sentence = "the bread is top notch as well"
14 _, pred, _ = predict(best_model, sentence, device) # => [0, 1, 0, 0, 0, 0, 0] => bread
```



# Phần III. Aspect Term Sentiment Classification

Trong phần 1, chúng ta đã tìm được từ là khía cạnh trong văn bản đầu vào, ở phần này, chúng ta xây dựng mô hình dựa vào câu đầu vào và khía cạnh được dự đoán xây dựng mô hình dự đoán cảm xúc.



Hình 4: Aspect term sentiment classification pipeline.

## 1. Build Dataset

```

1 # load dataset
2 import pandas as pd
3
4 train_df = pd.read_csv('./data/restaurants_train.csv')
5 test_df = pd.read_csv('./data/restaurants_test.csv')
6
7 # tokenizer
8 from transformers import BertTokenizer
9
10 model_name = "bert-base-uncased"
11 tokenizer = BertTokenizer.from_pretrained(model_name)
12
13 # build dataset
14 import torch
15 from torch.utils.data import Dataset
16
17 class ABSADataset(Dataset):
18     def __init__(self, df, tokenizer):
19         self.df = df
20         self.tokenizer = tokenizer
21
22     def __getitem__(self, idx):

```

```

23     tokens, tags, pols = self.df.iloc[idx, :3].values
24
25     tokens = tokens.replace("'", "").strip("[").split(',')
26     tags = tags.strip('][').split(',')
27     pols = pols.strip('][').split(',')
28
29     bert_tokens = []
30     bert_att = []
31     pols_label = 0
32     for i in range(len(tokens)):
33         t = self.tokenizer.tokenize(tokens[i])
34         bert_tokens += t
35         if int(pols[i]) != -1:
36             bert_att += t
37             pols_label = int(pols[i])
38
39     segment_tensor = [0] + [0]*len(bert_tokens) + [0] + [1]*len(bert_att)
40     bert_tokens = ['[CLS]'] + bert_tokens + ['[SEP]'] + bert_att
41
42
43     bert_ids = self.tokenizer.convert_tokens_to_ids(bert_tokens)
44
45     ids_tensor = torch.tensor(bert_ids)
46     pols_tensor = torch.tensor(pols_label)
47     segment_tensor = torch.tensor(segment_tensor)
48
49     return bert_tokens, ids_tensor, segment_tensor, pols_tensor
50
51     def __len__(self):
52         return len(self.df)
53
54 train_ds = ABSADataset(train_df, tokenizer)
55 test_ds = ABSADataset(test_df, tokenizer)
56
57 # dataloader
58 from torch.nn.utils.rnn import pad_sequence
59
60 def padding(samples):
61     ids_tensors = [s[1] for s in samples]
62     ids_tensors = pad_sequence(ids_tensors, batch_first=True)
63
64     segments_tensors = [s[2] for s in samples]
65     segments_tensors = pad_sequence(segments_tensors, batch_first=True)
66
67     label_ids = torch.stack([s[3] for s in samples])
68
69     masks_tensors = torch.zeros(ids_tensors.shape, dtype=torch.long)
70     masks_tensors = masks_tensors.masked_fill(ids_tensors != 0, 1)
71
72     return ids_tensors, segments_tensors, masks_tensors, label_ids
73
74 from torch.utils.data import DataLoader
75
76 batch_size = 32
77 train_loader = DataLoader(
78     train_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
79 )
80 test_loader = DataLoader(
81     test_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
82 )

```

## 2. Modeling

```
1 from transformers import BertModel
2
3 class ABSABert(torch.nn.Module):
4     def __init__(self, model_name):
5         super(ABSABert, self).__init__()
6         self.bert = BertModel.from_pretrained(model_name)
7         self.linear = torch.nn.Linear(self.bert.config.hidden_size, 3)
8         self.loss_fn = torch.nn.CrossEntropyLoss()
9
10    def forward(self, ids_tensors, masks_tensors, segments_tensors, lable_tensors):
11        out_dict = self.bert(
12            input_ids=ids_tensors,
13            attention_mask=masks_tensors,
14            token_type_ids=segments_tensors
15        )
16        linear_outputs = self.linear(out_dict['pooler_output'])
17
18        if lable_tensors is not None:
19            loss = self.loss_fn(linear_outputs, lable_tensors)
20            return loss, linear_outputs
21        else:
22            return linear_outputs
```

### 3. Trainer

```

1 import time
2 import numpy as np
3
4 def train_epoch(model, optimizer, train_loader, device):
5     losses = []
6     for batch in (train_loader):
7         ids_tensors, segments_tensors, masks_tensors, label_ids = batch
8         ids_tensors = ids_tensors.to(device)
9         segments_tensors = segments_tensors.to(device)
10        label_ids = label_ids.to(device)
11        masks_tensors = masks_tensors.to(device)
12
13        loss, _ = model(
14            ids_tensors=ids_tensors,
15            masks_tensors=masks_tensors,
16            segments_tensors=segments_tensors,
17            label_tensors=label_ids
18        )
19        losses.append(loss.item())
20        loss.backward()
21        optimizer.step()
22        optimizer.zero_grad()
23    return sum(losses)/len(losses)
24
25 def evaluate_epoch(model, valid_loader, device):
26     losses = []
27
28     preds, labels = [], []
29     with torch.no_grad():
30         for batch in (valid_loader):
31             ids_tensors, segments_tensors, masks_tensors, label_ids = batch
32             ids_tensors = ids_tensors.to(device)
33             segments_tensors = segments_tensors.to(device)
34             masks_tensors = masks_tensors.to(device)
35             label_ids = label_ids.to(device)
36
37             loss, outputs = model(
38                 ids_tensors=ids_tensors,
39                 masks_tensors=masks_tensors,
40                 segments_tensors=segments_tensors,
41                 label_tensors=label_ids
42             )
43             losses.append(loss.item())
44
45             _, p = torch.max(outputs, dim=1)
46             preds += list([int(i) for i in p])
47             labels += list([int(i) for i in label_ids])
48
49     acc = np.mean(np.array(preds) == np.array(labels))
50     return sum(losses)/len(losses), acc
51
52 def train(model, model_name, save_model, optimizer, train_loader, valid_loader,
53          num_epochs, device):
54     train_losses = []
55     eval_accs, eval_losses = [], []
56     best_loss_eval = 100
57     times = []
58     for epoch in range(1, num_epochs+1):

```

```

58     epoch_start_time = time.time()
59     # Training
60     train_loss = train_epoch(model, optimizer, train_loader, device)
61     train_losses.append(train_loss)
62
63     # Evaluation
64     eval_loss, eval_acc = evaluate_epoch(model, valid_loader, device)
65     eval_accs.append(eval_acc)
66     eval_losses.append(eval_loss)
67
68     # Save best model
69     if eval_loss < best_loss_eval:
70         torch.save(model.state_dict(), save_model + f'/{model_name}.pt')
71
72     times.append(time.time() - epoch_start_time)
73     # Print loss, acc end epoch
74     print("-" * 59)
75     print(
76         "| End of epoch {:3d} | Time: {:.2f}s | Train Loss {:.8f} "
77         "| Valid Accuracy {:.8f} | Valid Loss {:.8f} ".format(
78             epoch, time.time() - epoch_start_time, train_loss, eval_acc, eval_loss
79         )
80     )
81     print("-" * 59)
82
83     # Load best model
84     model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
85     model.eval()
86     metrics = {
87         'train_loss': train_losses,
88         'valid_accuracy': eval_accs,
89         'valid_loss': eval_losses,
90         'time': times
91     }
92     return model, metrics

```

#### 4. Training

```

1 save_model = "./model"
2 model = ABSABert(model_name)
3 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
4 model.to(device)
5 optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
6 num_epochs = 5
7 best_model, metrics = train(
8     model, model_name, save_model, optimizer, train_loader, test_loader, num_epochs,
9     device
10 )

```

Kết quả training và accuracy trên tập test là 81.4

End of epoch	1	Time: 46.77s	Train Loss	0.829	Valid Accuracy	0.748	Valid Loss	0.644
End of epoch	2	Time: 39.83s	Train Loss	0.539	Valid Accuracy	0.801	Valid Loss	0.507
End of epoch	3	Time: 39.13s	Train Loss	0.353	Valid Accuracy	0.814	Valid Loss	0.469
End of epoch	4	Time: 40.16s	Train Loss	0.218	Valid Accuracy	0.777	Valid Loss	0.623
End of epoch	5	Time: 39.52s	Train Loss	0.155	Valid Accuracy	0.805	Valid Loss	0.587

Hình 5: Quá trình huấn luyện ATSC.

## 5. Prediction

```

1 def predict(model, tokenizer, sentence, aspect, device):
2     t1 = tokenizer.tokenize(sentence)
3     t2 = tokenizer.tokenize(aspect)
4
5     word_pieces = ['[CLS]'] + t1 + ['[SEP]'] + t2
6
7     segment_tensor = [0] + [0]*len(t1) + [0] + [1]*len(t2)
8
9     input_ids = tokenizer.convert_tokens_to_ids(word_pieces)
10    input_tensor = torch.tensor([input_ids]).to(device)
11    segment_tensor = torch.tensor([segment_tensor]).to(device)
12
13    with torch.no_grad():
14        outputs = model(input_tensor, None, segment_tensor, None)
15        _, predictions = torch.max(outputs, dim=1)
16
17    return word_pieces, int(predictions), outputs
18
19 sentence = "The bread is top notch as well"
20 aspect = "bread"
21 predict(best_model, tokenizer, sentence, aspect, device) # => 2

```

## Phần 4. Câu hỏi trắc nghiệm

Cho câu bình luận như sau: "The food is very fresh."

**Câu hỏi 1** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ATE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 2** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ASPE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 3** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOPE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 4** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOSTE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 5** Mô hình giải quyết bài toán ATE ở phần 2 trên sử dụng bộ dữ liệu nào?

- a) IMDB-Review
- b) Penn Tree Bank
- c) SemEval-2014 Task 4
- d) SQuAD

**Câu hỏi 6** Số lượng samples được sử dụng cho tập train trong phần thực nghiệm là

- a) 3600
- b) 3601
- c) 3602
- d) 3603

**Câu hỏi 7** Mô hình giải quyết bài toán ATE ở phần 2 trên dựa vào bài toán tổng quát nào sau đây?

- a) Sequence Labeling
- b) Machine Translation

- c) Topic Modeling
- d) Sumarization

**Câu hỏi 8** Mô hình tiền huấn luyện sử dụng cho bài toán ATE ở phần 2 trên là?

- a) BERT
- b) RoBERTA
- c) DEBERTA
- d) DistilBERT

**Câu hỏi 9** Mô hình tiền huấn luyện sử dụng cho bài toán ASTC ở phần 3 trên là?

- a) BERT
- b) RoBERTA
- c) DEBERTA
- d) DistilBERT

**Câu hỏi 10** Hàm mục tiêu để giải quyết bài toán ASTC ở phần 3 là

- a) Masked Language Model
- b) Next Sentence Prediction
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

- *Hết* -