

+ New chat

Model: GPT-4

网页保存为PDF

基于BERT的SQL判断模型

RL Trine Training.

Trine Game Introduction

McAfee白名单机制

获取Street Fighter ROM

Python 贪吃蛇游戏

Python贪吃蛇游戏

Win10快捷方式图标问题

Fish and Water Quality.

YOLO for fishery detection.

HTML设置Arial字体

汪志鹏

...

< 2 / 2 >



输入是句子和table格式，输出是1和0判断是否这句话可以在这个table上转换成sql语句，写一个基于bert的微调模型，用huggingface和pytorch



要构建一个基于BERT的微调模型，用于判断输入句子和表格是否可以转换成SQL语句，我们可以使用Hugging Face和PyTorch。以下是实现这个任务的步骤：

1. 安装必要的库：

bash

Copy code

```
pip install transformers torch
```

2. 导入所需的库和模块：

python

Copy code

```
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, Train
```

3. 准备数据集：

定义一个自定义的Dataset类，接收文本句子、表格数据和标签（1或0）。在这里，我们将句子和表格数据连接在一起，用特殊的分隔符隔开。

python

Copy code

```
class SQLDataset(Dataset):
    def __init__(self, sentences, tables, labels, tokenizer, max_length):
        self.sentences = sentences
        self.tables = tables
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        table = self.tables[idx]
        label = self.labels[idx]
        inputs = self.tokenizer(
            f'{sentence} [SEP] {table}',
            return_tensors='pt',
            max_length=self.max_length,
            padding='max_length',
            truncation=True
        )
        inputs['labels'] = torch.tensor(label, dtype=torch.long)
        return inputs
```

4. 创建一个训练参数的实例：

python

Copy code

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy='epoch',
    save_strategy='epoch',
    seed=42,
    load_best_model_at_end=True,
)
```

5. 加载预训练的BERT模型并微调：

python

Copy code

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", n

# 将句子、表格和标签列表传递给SQLDataset实例
train_dataset = SQLDataset(sentences, tables, labels, tokenizer, max_length=
train_dataloader = DataLoader(train_dataset, batch_size=16)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

trainer.train()
```

6. 评估和使用模型:

要评估模型，可以使用Trainer的`evaluate`方法。为了使用模型对新的输入进行预测，只需将新的句子和表格传递给模型即可。

```
python Copy code

# 评估模型
trainer.evaluate()

# 使用模型进行预测
sentence = "Example sentence"
table = "Example table"
inputs = tokenizer(f"{sentence} [SEP] {table}", return_tensors="pt", max_len
outputs = model(*inputs)
prediction = torch.argmax(outputs.logits, dim=-1).item()
```

根据这些步骤，您可以创建一个基于BERT的微调模型



我写得读取数据：df = pd.read_csv("./data/data.csv") sentences = df['Input'].values
tables = df['Table'].values
labels = df['Output'].values
报错：Output exceeds the size limit. Open the full output data in a text editor-----

```
-----  
ValueError Traceback (most recent call last)  
Cell In[23], line 14  
6 train_dataloader = DataLoader(train_dataset, batch_size=16)  
8 trainer = Trainer(  
9     model=model,  
10    args=training_args,  
11    train_dataset=train_dataset,  
12)  
---> 14 trainer.train()
```

```
File d:\anaconda3\envs\huggingface\lib\site-packages\transformers\trainer.py:1633, in
Trainer.train(self, resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
1628     self.model_wrapped = self.model
1630     inner_training_loop = find_executable_batch_size(
1631         self._inner_training_loop, self._train_batch_size, args.auto_find_batch_size
1632     )
-> 1633     return inner_training_loop(
1634         args=args,
1635         resume_from_checkpoint=resume_from_checkpoint,
1636         trial=trial,
1637         ignore_keys_for_eval=ignore_keys_for_eval,
1638     )
```

```
File d:\anaconda3\envs\huggingface\lib\site-packages\transformers\trainer.py:1902, in
Trainer._inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial,
ignore_keys_for_eval)
1900     tr_loss_step = self.training_step(model, inputs)
...
--> 974 batch_size, seq_length = input_shape
975 device = input_ids.device if input_ids is not None else inputs_embeds.device
977 # past_key_values_length
```

ValueError: too many values to unpack (expected 2)



根据您提供的错误信息，问题可能出在自定义的`SQLDataset`类的`__getitem__`方法。在`'inputs'`字典中，需要将每个输入值的第一维度压缩为1。修改`__getitem__`方法如下：

```
python Copy code
```

```
def __getitem__(self, idx):
    sentence = self.sentences[idx]
    table = self.tables[idx]
    label = self.labels[idx]
    inputs = self.tokenizer(
        f'{sentence} [SEP] {table}',
        return_tensors="pt",
        max_length=self.max_length,
        padding="max_length",
        truncation=True
    )
    inputs = {key: value.squeeze(0) for key, value in inputs.items()} # Add
    inputs['labels'] = torch.tensor(label, dtype=torch.long)
    return inputs
```

这个修改应该可以解决`ValueError: too many values to unpack (expected 2)`问题。
现在，您应该能够正确地训练模型。



Output exceeds the size limit. Open the full output data in a text editor-----

ValueError Traceback (most recent call last)

Cell In[25], line 14

```
6 train_dataloader = DataLoader(train_dataset, batch_size=16)
8 trainer = Trainer(
9     model=model,
10    args=training_args,
11    train_dataset=train_dataset,
12)
---> 14 trainer.train()
```

```
File d:\anaconda3\envs\huggingface\lib\site-packages\transformers\trainer.py:1633, in
Trainer.train(self, resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
1628     self.model_wrapped = self.model
1630 inner_training_loop = find_executable_batch_size(
1631     self._inner_training_loop, self._train_batch_size, args.auto_find_batch_size
1632)
-> 1633 return inner_training_loop(
1634     args=args,
1635     resume_from_checkpoint=resume_from_checkpoint,
1636     trial=trial,
1637     ignore_keys_for_eval=ignore_keys_for_eval,
1638)
```

```
File d:\anaconda3\envs\huggingface\lib\site-packages\transformers\trainer.py:1994, in
Trainer._inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial,
ignore_keys_for_eval)
1991     self.control.should_training_stop = True
...
--> 935 raise ValueError("Trainer: evaluation requires an eval_dataset.")
936 eval_dataset = eval_dataset if eval_dataset is not None else self.eval_dataset
937 data_collator = self.data_collator
```

ValueError: Trainer: evaluation requires an eval_dataset.



根据您提供的错误信息，问题是在训练过程中需要一个验证集（`eval_dataset`），但在创建`Trainer`实例时没有提供。您可以选择使用单独的验证集或在数据集中划分一部分作为验证集。以下是一个使用训练集和验证集的例子：

首先，创建一个`SQLDataset`实例并将其分为训练集和验证集：

```
python
from sklearn.model_selection import train_test_split

# 将数据分为训练集和验证集
sentences_train, sentences_val, tables_train, tables_val, labels_train, labels_val = train_test_split(
    sentences, tables, labels, test_size=0.2, random_state=42
)

train_dataset = SQLDataset(sentences_train, tables_train, labels_train, tokenizer, max_length)
val_dataset = SQLDataset(sentences_val, tables_val, labels_val, tokenizer, max_length)
```

然后，在创建`Trainer`实例时，将验证集传递给`eval_dataset`参数：

```
python
trainer = Trainer(
```

```
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset # Add this line
    )

trainer.train()
```

现在，您应该能够在每个训练阶段结束时执行验证。这将解决``ValueError: Trainer: evaluation requires an eval_dataset``问题。

我该怎么用训练完的模型？

训练完成后，您可以使用训练好的模型进行预测。以下是如何使用训练好的模型对新输入（句子和表格）进行预测的示例：

```
python
def predict(sentence, table, model, tokenizer, max_length=128):
    inputs = tokenizer(
        f"[sentence] [SEP] {table}",
        return_tensors="pt",
        max_length=max_length,
        padding="max_length",
        truncation=True
    )

    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        prediction = torch.argmax(logits, dim=-1).item()

    return prediction

# 示例输入
sentence = "Example sentence"
table = "Example table"

# 预测
prediction = predict(sentence, table, model, tokenizer)

# 输出预测结果
if prediction == 1:
    print("The input sentence can be converted into an SQL query using the p
else:
    print("The input sentence cannot be converted into an SQL query using th
```

在这个示例中，我们定义了一个名为`predict`的函数，它接受一个句子、一个表格、一个模型和一个分词器作为参数。该函数首先使用分词器将输入句子和表格编码为模型需要的格式，然后使用模型进行预测。最后，我们根据预测结果输出相应的信息。

您可以将这个`predict`函数用于您自己的输入数据来测试训练好的模型。

RuntimeError Traceback (most recent call last)

```
Cell In[28], line 22
 19 table = "Example table"
 20 # 预测
--> 21 prediction = predict(sentence, table, model, tokenizer)
 22 # 输出预测结果
 23 if prediction == 1:
```

```
Cell In[28], line 11, in predict(sentence, table, model, tokenizer, max_length)
  2 inputs = tokenizer(
  3     f"[sentence] [SEP] {table}",
  4     return_tensors="pt",
  5     ...
  6     truncation=True
  7 )
 8
10 with torch.no_grad():
--> 11     outputs = model(**inputs)
 12     logits = outputs.logits
 13     prediction = torch.argmax(logits, dim=-1).item()
```

```
File d:\anaconda3\envs\huggingface\lib\site-packages\torch\nn\modules\module.py:1501,
in Module__call__(self, *args, **kwargs)
1496 # If we don't have any hooks, we want to skip the rest of the logic in
```

```
1497 # this function, and just call forward.  
1498 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or  
self._forward_pre_hooks  
...  
2208 # remove once script supports set_grad_enabled  
2209 _no_grad_embedding_renorm_(weight, input, max_norm, norm_type)  
-> 2210 return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)  
  
RuntimeError: Expected all tensors to be on the same device, but found at least two devices,  
cuda:0 and cpu! (when checking argument index in method  
wrapper_CUDA_index_select)
```

这个错误是由于模型和输入数据在不同的设备（CPU和GPU）上。要解决此问题，需要确保 模型和输入数据都在相同的设备上。您可以使用以下代码来实现这一点：

首先，检测是否有可用的GPU：

```
python  
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

接着，将模型移动到相应的设备：

```
python  
model.to(device)
```

最后，确保在`predict`函数中将输入数据也移动到相同的设备：

```
python  
def predict(sentence, table, model, tokenizer, device, max_length=128):  
    inputs = tokenizer(  
        f"{sentence} [SEP] {table}",  
        return_tensors="pt",  
        max_length=max_length,  
        padding="max_length",  
        truncation=True  
    )  
  
    # 将输入数据移动到相同的设备  
    inputs = {key: value.to(device) for key, value in inputs.items()}  
  
    with torch.no_grad():  
        outputs = model(**inputs)  
        logits = outputs.logits  
        prediction = torch.argmax(logits, dim=-1).item()  
  
    return prediction
```

现在，您应该可以正常使用训练好的模型进行预测。请注意，如果您的设备是GPU，这将在GPU上执行预测；否则，将在CPU上执行预测。

最后我想保存模型

要保存训练好的模型，您可以使用Hugging Face Transformers库提供的`save_pretrained`方法。以下是如何保存模型和分词器到指定目录的示例：

```
python  
# 指定保存目录  
save_directory = "path/to/save/directory"  
  
# 保存模型  
model.save_pretrained(save_directory)  
  
# 保存分词器  
tokenizer.save_pretrained(save_directory)
```

现在，您的模型和分词器将被保存到指定的目录。以后，您可以使用`from_pretrained`方法从该目录加载模型和分词器进行使用。下面是如何加载已保存的模型和分词器的示例：

```
from transformers import BertForSequenceClassification, BertTokenizer

# 加载已保存的模型
loaded_model = BertForSequenceClassification.from_pretrained(save_directory)

# 加载已保存的分词器
loaded_tokenizer = BertTokenizer.from_pretrained(save_directory)

# 如果需要，将模型移动到设备上
loaded_model.to(device)
```

现在，您可以使用加载的模型和分词器进行预测或进一步训练。