山东大学＿＿计算机科学与技术＿＿学院

＿＿大数据分析与实践＿课程实验报告

学号：202300130242	姓名：王启源	班级：数据 23
实验题目：实验 4		
实验学时：2	实验日期：2025.12.18	

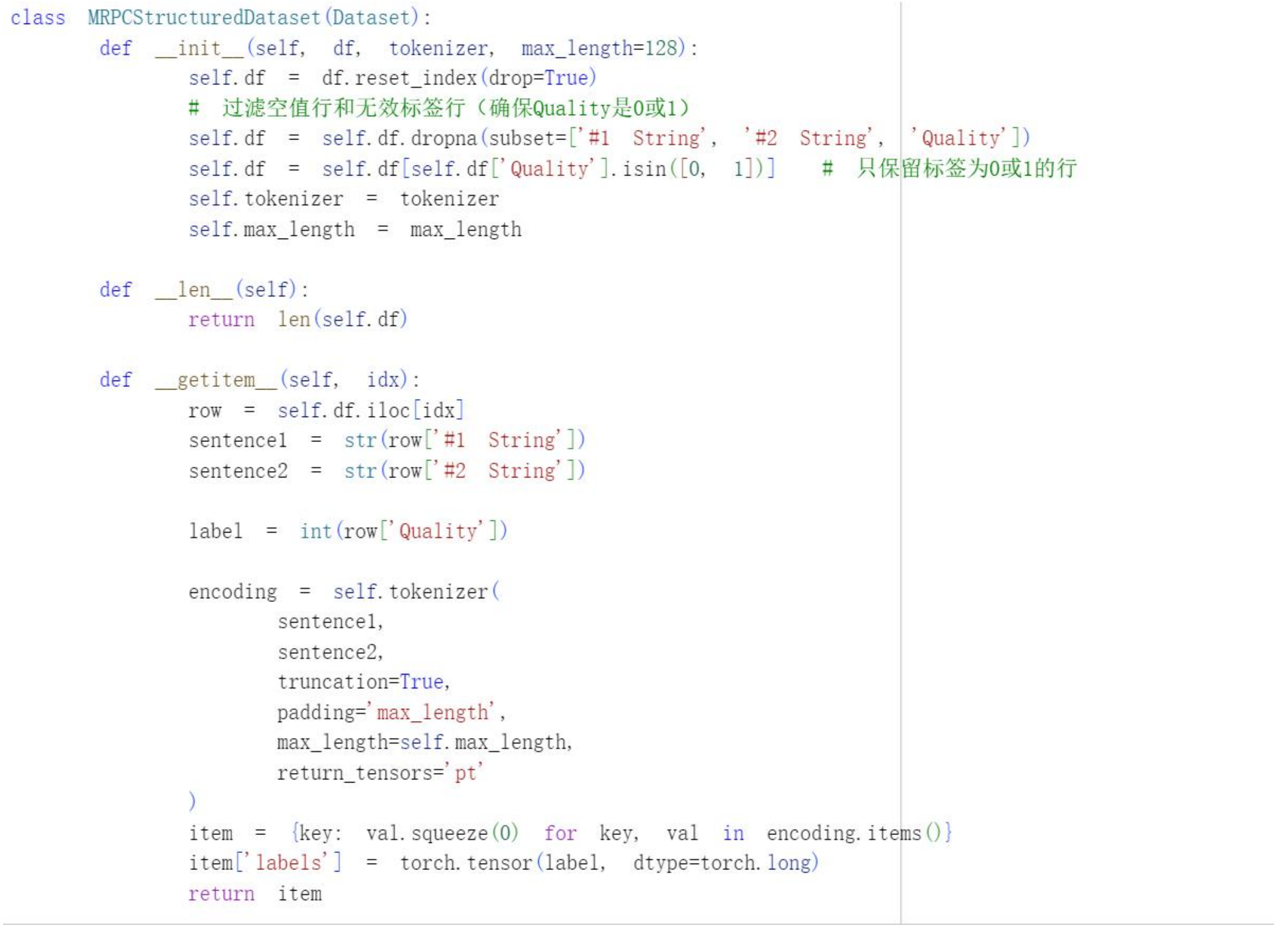
实验目的：
训练 bert 模型，使其适用于句子一致性判断问题

硬件环境：
计算机一台

软件环境：
Linux 或 Windows

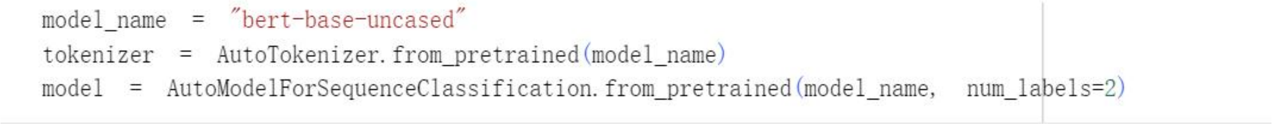
实验步骤与内容：

1.自定义一个数据集，这个数据集适用于 MRPC 数据集



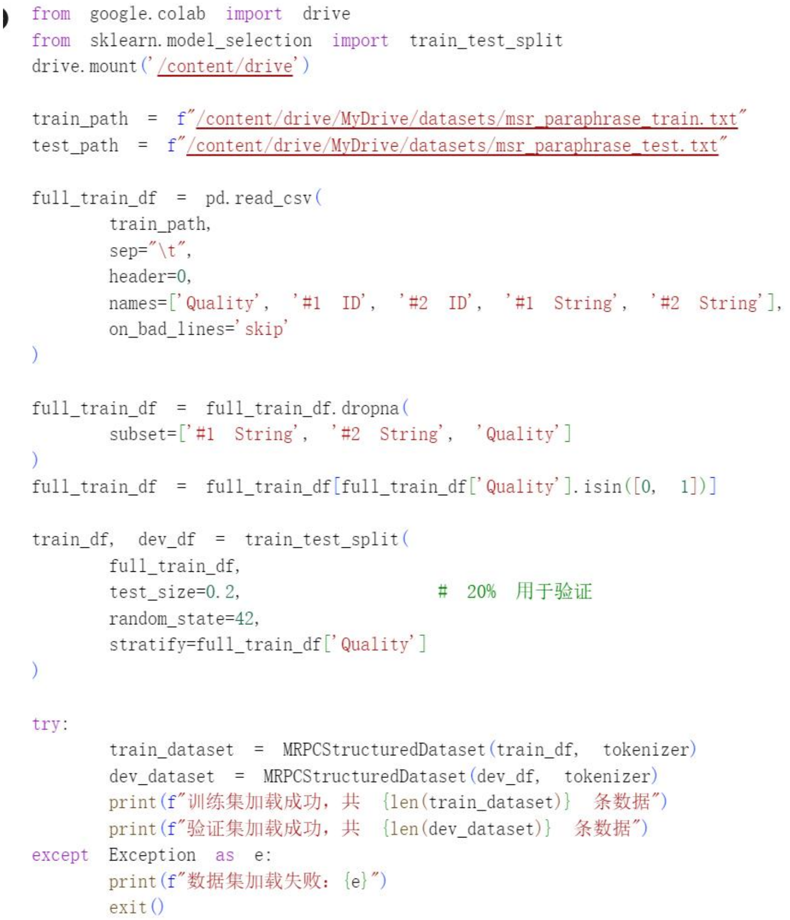
```
class MRPCStructuredDataset(Dataset):
    def __init__(self, df, tokenizer, max_length=128):
        self.df = df.reset_index(drop=True)
        # 过滤空值行和无效标签行（确保Quality是0或1）
        self.df = self.df.dropna(subset=['#1 String', '#2 String', 'Quality'])
        self.df = self.df[self.df['Quality'].isin([0, 1])]   # 只保留标签为0或1的行
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        sentence1 = str(row['#1 String'])
        sentence2 = str(row['#2 String'])

        label = int(row['Quality'])

        encoding = self.tokenizer(
            sentence1,
            sentence2,
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors='pt'
        )
        item = {key: val.squeeze(0) for key, val in encoding.items()}
        item['labels'] = torch.tensor(label, dtype=torch.long)
        return item
```

2.
导入预训练好的分词器和 bert 模型



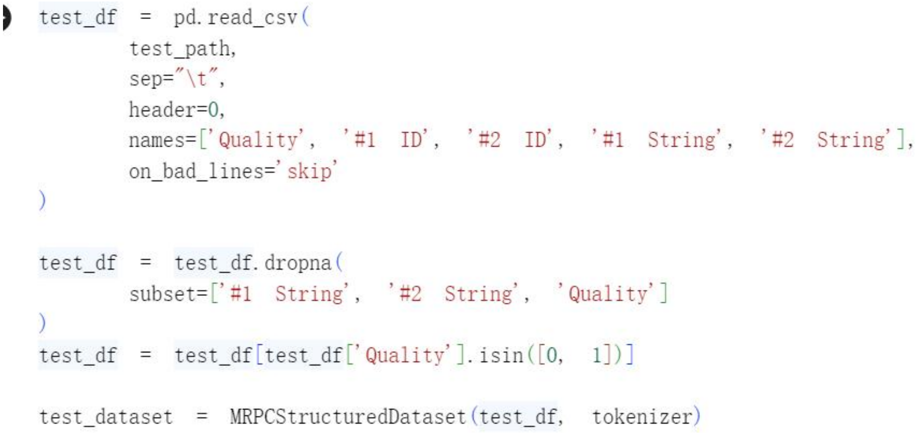
```python
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

3.将 MRPC 数据集导入，然后划分为训练集和验证集



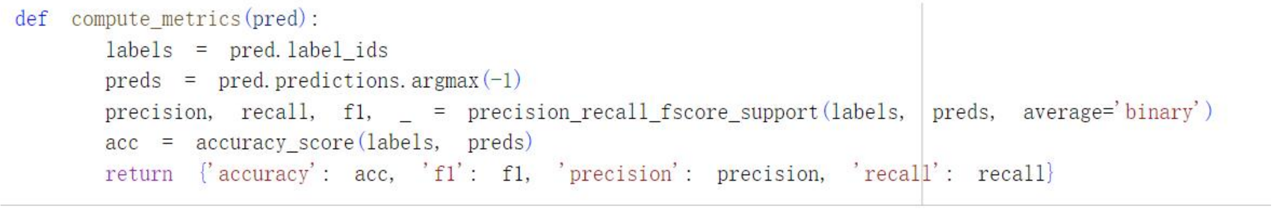
```python
from google.colab import drive
from sklearn.model_selection import train_test_split
drive.mount('/content/drive')

train_path = f"/content/drive/MyDrive/datasets/msr_paraphrase_train.txt"
test_path = f"/content/drive/MyDrive/datasets/msr_paraphrase_test.txt"

full_train_df = pd.read_csv(
        train_path,
        sep="\t",
        header=0,
        names=['Quality', '#1 ID', '#2 ID', '#1 String', '#2 String'],
        on_bad_lines='skip'
)

full_train_df = full_train_df.dropna(
        subset=['#1 String', '#2 String', 'Quality']
)
full_train_df = full_train_df[full_train_df['Quality'].isin([0, 1])]

train_df, dev_df = train_test_split(
        full_train_df,
        test_size=0.2,                    # 20% 用于验证
        random_state=42,
        stratify=full_train_df['Quality']
)

try:
        train_dataset = MRPCStructuredDataset(train_df, tokenizer)
        dev_dataset = MRPCStructuredDataset(dev_df, tokenizer)
        print(f"训练集加载成功，共 {len(train_dataset)} 条数据")
        print(f"验证集加载成功，共 {len(dev_dataset)} 条数据")
except Exception as e:
        print(f"数据集加载失败：{e}")
        exit()
```

4.导入测试集



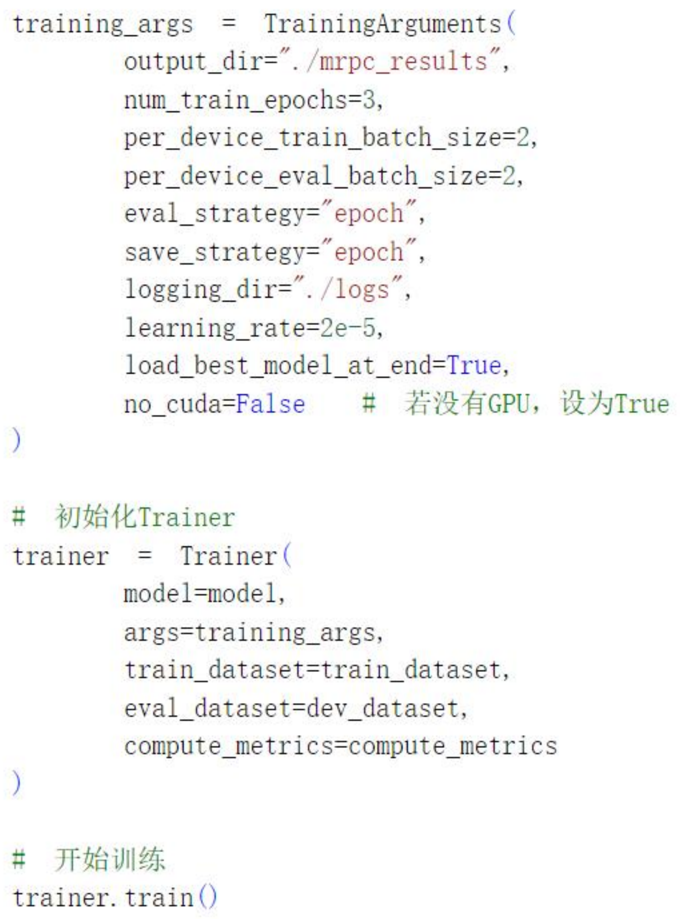
```python
test_df = pd.read_csv(
        test_path,
        sep="\t",
        header=0,
        names=['Quality', '#1 ID', '#2 ID', '#1 String', '#2 String'],
        on_bad_lines='skip'
)

test_df = test_df.dropna(
        subset=['#1 String', '#2 String', 'Quality']
)
test_df = test_df[test_df['Quality'].isin([0, 1])]

test_dataset = MRPCStructuredDataset(test_df, tokenizer)
```

5.该函数指定了在模型训练过程中会计算哪一些指标的分数

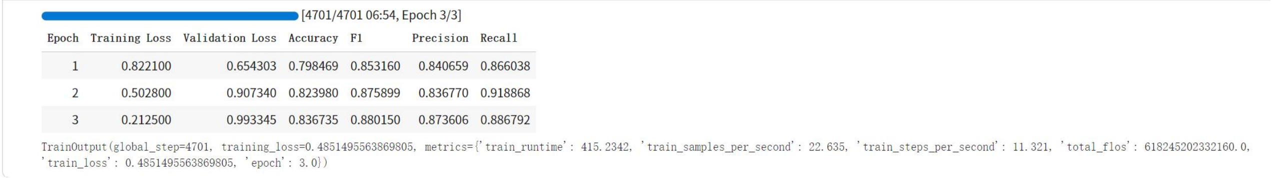


```
def  compute_metrics(pred):
        labels  =  pred.label_ids
        preds  =  pred.predictions.argmax(-1)
        precision,  recall,  f1,  _  =  precision_recall_fscore_support(labels,  preds,  average='binary')
        acc  =  accuracy_score(labels,  preds)
        return  {'accuracy':  acc,  'f1':  f1,  'precision':  precision,  'recall':  recall}
```

6.设定训练参数并开始训练

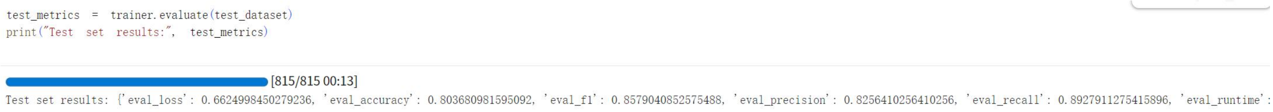


```
training_args  =  TrainingArguments(
        output_dir=″./mrpc_results″,
        num_train_epochs=3,
        per_device_train_batch_size=2,
        per_device_eval_batch_size=2,
        eval_strategy=″epoch″,
        save_strategy=″epoch″,
        logging_dir=″./logs″,
        learning_rate=2e-5,
        load_best_model_at_end=True,
        no_cuda=False    #  若没有GPU，设为True
)

#  初始化Trainer
trainer  =  Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=dev_dataset,
        compute_metrics=compute_metrics
)

#  开始训练
trainer.train()
```



```
[4701/4701 06:54, Epoch 3/3]
Epoch  Training Loss  Validation Loss  Accuracy  F1        Precision  Recall
    1       0.822100         0.654303  0.798469  0.853160  0.840659   0.866038
    2       0.502800         0.907340  0.823980  0.875899  0.836770   0.918868
    3       0.212500         0.993345  0.836735  0.880150  0.873606   0.886792
TrainOutput(global_step=4701, training_loss=0.4851495563869805, metrics={'train_runtime': 415.2342, 'train_samples_per_second': 22.635, 'train_steps_per_second': 11.321, 'total_flos': 618245202332160.0,
'train_loss': 0.4851495563869805, 'epoch': 3.0})
```

7.对测试集进行测试，得到最后的测试分数



```
test_metrics  =  trainer.evaluate(test_dataset)
print(″Test  set  results:″,  test_metrics)
```

```
[815/815 00:13]
Test set results: {'eval_loss': 0.6624998450279236, 'eval_accuracy': 0.803680981595092, 'eval_f1': 0.8579040852575488, 'eval_precision': 0.8256410256410256, 'eval_recall': 0.8927911275415896, 'eval_runtime':
```

结论分析与体会：
掌握了如何对 bert 模型进行一个微调，使其适用于其他的任务