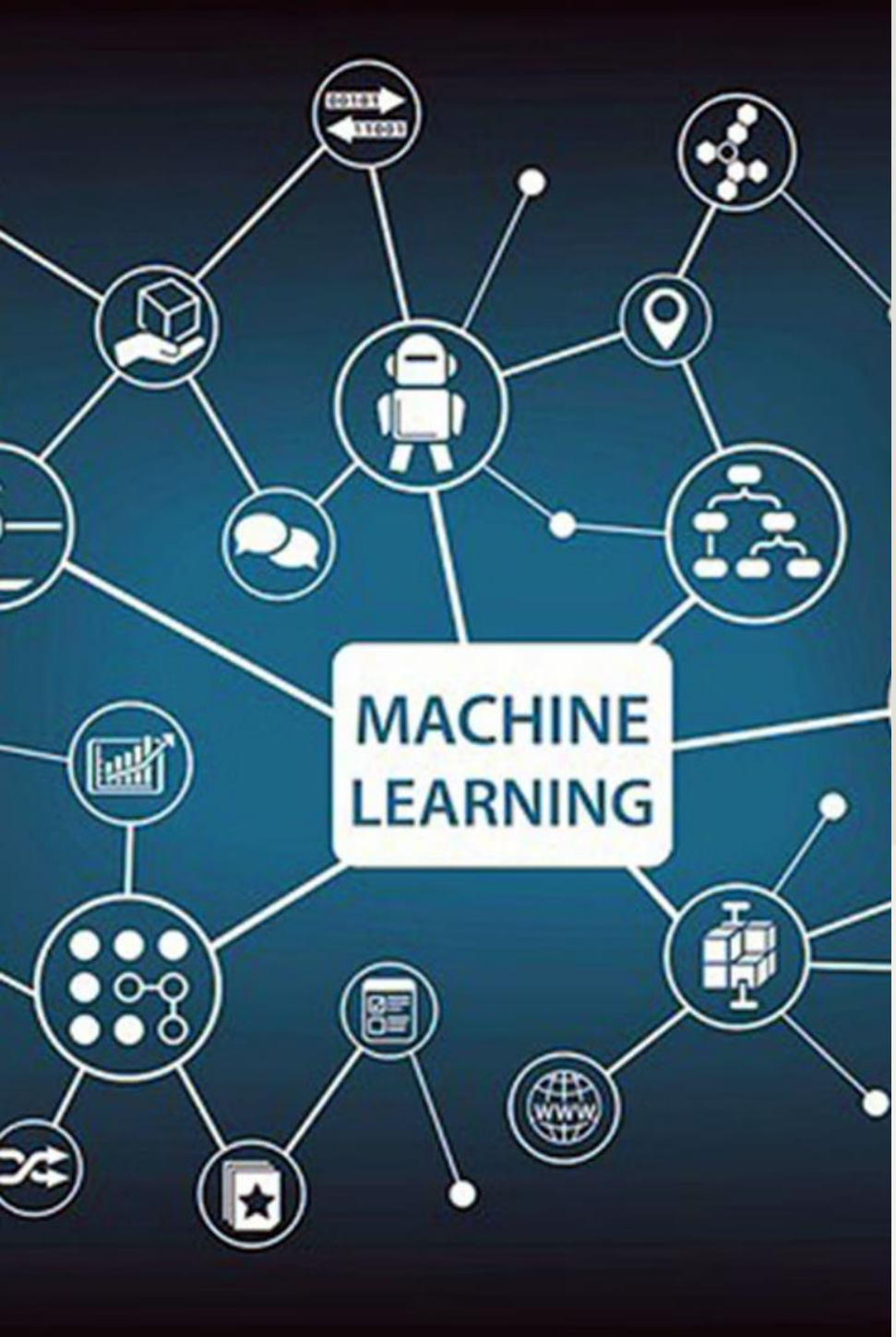

코드 유사성 판단 시즌2
AI 경진대회

[Private 3위/0.9861]- GraphcodeBert

code7ssage / 고려대학교 생명공학부 이승현

목차

1. 문제 인식

대회 주제, 고려할 점

대회 주제 : C++ 코드의 유사성 판단

평가지표: Accuracy

고려할 점:

1) train 코드들 살펴보니 길이가 긴 편 → 공백, 주석 등을 제거하는 전처리 필요

2) test 데이터에 비해 sample_train데이터의 양이 매우 부족함 → 새로운 train 데이터 필요

3) 코드의 중요한 내용은 뒷 부분에 주로 분포함 → tokenizer.truncation_side = 'left' 사용

2. 개발환경

1) colab 사용

- colab pro의 A100 40GB 사용 → batch size 고려해서 적어도 VRam 32GB이상 필요
- train code, data, submission 파일은 google_drive에 저장하고 사용 (drive.mount 필요)
- 라이브러리 설치 및 버전 확인

라이브러리	버전
1. pandas	1 .5 .3
2. numpy	1 .25 .2
3.torch	2 .2. 1+cu121
4.transformers	4 .38 .2
5.sklearn	1 .2 .2

2. 개발환경

2) 라이브러리 import, seed 고정

- private score 재현을 위해 가능한 많은 seed를 고정
- torch.set_float32_matmul_precision('high')→ PyTorch 연산의 정밀도를 설정 ("파이썬초보만" 님의 SBERT baseline에서 참고)

```
!pip install transformers torch pandas tqdm

import pandas as pd
import numpy as np
import os
import random
from itertools import combinations, product
import re
import sklearn
from sklearn.model_selection import train_test_split
import torch
torch.set_float32_matmul_precision('high')
from torch.utils.data import Dataset, DataLoader
import transformers
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW
from tqdm import tqdm

import warnings
warnings.filterwarnings('ignore')
from google.colab import files
```

```
# Seed 고정 함수
def seed_everything(seed: int = 42, contain_cuda: bool = False):
  os.environ['PYTHONHASHSEED'] = str(seed)
  random.seed(seed)
  np.random.seed(seed)

  torch.backends.cudnn.deterministic = True
  torch.backends.cudnn.benchmark = False

  torch.manual_seed(seed)
  torch.cuda.manual_seed(seed)
  torch.cuda.manual_seed_all(seed)
  print(f"Seed set as {seed}")

seed = 42
seed_everything(seed)
```

3. 데이터 생성

1) 코드 파일 읽기 및 데이터프레임 생성

- train_code에 500문제, 그리고 각 문제에 대한 solution 코드 500개가 존재
- 각 문제 폴더를 순회하면서 C++ 코드 파일의 내용을 읽고, 전처리 한 코드의 내용(`preprocess_scripts`)과 해당 문제 번호(`problem_nums`)를 리스트에 저장. 이후 이 리스트들을 사용하여 데이터프레임을 생성
- 폴더 안에 .ipynb_checkpoints 폴더는 solution 코드가 아님으로 무시

```
preprocess_scripts = []
problem_nums = []

# 500개 Sample code에 대한 전처리
for problem_folder in tqdm(problem_folders):
    scripts = os.listdir(os.path.join(code_folder, problem_folder)) # code/problem000/.cpp 파일
    problem_num = problem_folder # 문제 번호 폴더명
    for script in scripts:
        # .ipynb_checkpoints 폴더 무시
        if script == ".ipynb_checkpoints":
            continue
        script_file = os.path.join(code_folder,problem_folder,script)
        if os.path.isfile(script_file):  # 경로가 실제 파일인지 확인
            preprocessed_script =  preprocess_and_remove_extras(script_file)
            if preprocessed_script:  # 전처리된 스크립트가 비어 있지 않다면
                preprocess_scripts.append(preprocessed_script)
                problem_nums.append(problem_folder)  # 문제 번호 추가
```

3. 데이터 생성

2) train_data, valid_data 생성

- train_code로 만든 dataframe을 9:1로 분리
- 분리 한 df에서 각 문제당 Random하게 뽑은 Positive pairs와 Negative pairs로 새 df를 만듦
- sample_train이 중요한 정보를 가지고 있을 수 있으니, 생성한 df에 통합해서 사용
- 데이터 생성 방식은 코드 유사성 시즌 1 "나일강"님의 코드 참조
- 첫번째 모델은 positive pairs + negative pairs = 500 x 1500 + 500 x 1500 = 1500000 (150만 pairs),
  두번째 모델은 positive pairs + negative pairs = 500 x 2000 + 500 x 2000 = 1500000 (200만 pairs)
  에다가 각각 sample_train 20000pairs를 더한 train_data를 사용
- valid 데이터는 두 모델 다 20만 pairs 사용

3. 데이터 생성

2) train_data, valid_data 생성

- 첫번째 모델의 train_data 형성만 보여줌, 나머지 코드도 pair 수만 다르고 동일

```
codes = train_df['code'].to_list() # code 컬럼을 list로 변환 - codes는 code가 쭉 나열된 형태임
problems = train_df['problem_num'].unique().tolist() # 문제 번호를 중복을 제외하고 list로 변환
problems.sort()

train_positive_pairs = []
train_negative_pairs = []

for problem in tqdm(problems):
    # 각각의 문제에 대한 code를 골라 정답 코드로 저장, 아닌 문제는 other_codes로 저장
    # 이때 train_df에는 problem_num이 정렬된 상태가 아니기 때문에 index가 다를 수 있음
    solution_codes = train_df[train_df['problem_num'] == problem]['code'].to_list()
    other_codes = train_df[train_df['problem_num'] != problem]['code'].to_list()

    # positive_pairs 1500개 (총 500 * 1500 = 750,000개) 추출
    # negative_pairs 1500개 (총 500 * 1500 = 750,000개) 추출
    positive_pairs = list(combinations(solution_codes,2))
    random.shuffle(positive_pairs)
    positive_pairs = positive_pairs[:1500]
    random.shuffle(other_codes)
    other_codes = other_codes[:1500]

    negative_pairs = []
    for pos_codes, others in zip(positive_pairs, other_codes):
        negative_pairs.append((pos_codes[0], others))

    train_positive_pairs.extend(positive_pairs)
    train_negative_pairs.extend(negative_pairs)
```

4. 모델링

1) 전처리

- create_data는 아까 생성한 train_df, 대회에서 주어진 sample 데이터와 합쳐 사용
- 나중에 model의 max_length를 512로 설정 할 것을 감안하면 코드의 길이가 너무 긺 → 길이를 줄이는데 초점을 맞춘 전처리 필요

1)-1 주석제거: 싱글라인, 멀티라인 둘 다 제거

1)-2 #inclue 지시문 제거: angle brackets, double quotes 둘 다 제거

1)-3 #define 제거: 매크로 정의 제거

1)-4 공백, 줄바꿈 축소: 탭과 여러 공백을 하나의 공백으로, 여러 줄바꿈을 하나로

4. 모델링

1) 전처리- 코드

```
create_train_df = pd.read_csv(create_train_data_path)
sample_df = pd.read_csv(sample_data_path)
train_df = pd.concat([create_train_df, sample_df], ignore_index=True) # 두 학습데이터를 결합
val_df = pd.read_csv(valid_data_path)

#  전처리 적용
def remove_extras(code):
    code = re.sub(re.compile("/\*.*?\*/", re.DOTALL), "", code) # 멀티 라인 주석 제거
    code = re.sub(re.compile("//.*?\n"), "", code) # 싱글 라인 주석 제거
    code = re.sub(re.compile("#include <.*?>\n"), "", code)  # angle brackets를 사용하는 include 제거
    code = re.sub(re.compile("#include \".*?\"\n"), "", code)  # double quotes를 사용하는 include 제거
    code = re.sub(re.compile("#define .*?\n"), "", code)  # 매크로 정의 제거
    code = re.sub(re.compile("[\t ]+"), " ", code)  # 탭과 여러 공백을 하나의 공백으로
    code = re.sub(re.compile("\n\s*\n"), "\n", code)  # 여러 줄바꿈을 하나로

    return code.strip()
```

4. 모델링

1) 전처리 - 데이터셋 class 정의

- train, valid, test 전부 같은 데이터셋 형식 적용

```python
# 데이터셋 클래스 정의
class CodePairDataset(Dataset):
    def __init__(self, tokenizer, data, max_length=512, include_labels=True):
        self.tokenizer = tokenizer
        self.data = data  # 이제 data는 DataFrame 객체입니다.
        self.max_length = max_length
        self.include_labels = include_labels

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

    def __getitem__(self, idx):
        record = self.data.iloc[idx]
        code1 = remove_extras(record['code1'])
        code2 = remove_extras(record['code2'])

        inputs = self.tokenizer(
            code1, code2,
            padding='max_length', truncation=True, max_length=self.max_length, return_tensors="pt"
        )
        inputs = {key: val.squeeze() for key, val in inputs.items()}
        if self.include_labels:
            inputs['labels'] = torch.tensor(record['similar'], dtype=torch.long)
        return inputs
```

4. 모델링

2) 모델 설계

- 모델은 graphcodebert로 고정
- 대신 train 데이터를 두 종류 만들어서 모델 train후, 모델 저장 (152만쌍, 202만쌍)
- 최종모델은 각 모델을 평가 모드로 설정하고, 테스트 데이터셋에 대해 예측을 수행
- 각 데이터 포인트에 대해 모델의 출력을 softmax 함수에 통과시켜, 각 클래스에 속할 확률을 구함
- 두 모델의 예측 확률을 평균내어 최종 예측 확률을 구함

** 각 모델은 train 데이터, epoch 수를 제외하고는 다 동일하게 유지해서 일관성을 유지함
ex) batch size =48, max_length = 512는, learning rate = 2e-5 유지

4. 모델링

3) 모델링 코드- 첫번째 모델 train 152만쌍, epoch수 4

```
# GraphCodeBERT 모델 및 토크나이저 로드
model_name = "microsoft/graphcodebert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.truncation_side = 'left'
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2).to(device)


# 훈련 세트와 검증 세트에 대한 데이터셋 생성
train_dataset = CodePairDataset(tokenizer, train_df, max_length=512)
val_dataset = CodePairDataset(tokenizer, val_df, max_length=512, include_labels=True)

# 데이터 로더 준비
train_loader = DataLoader(train_dataset, batch_size=48, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=48, shuffle=False)

# 옵티마이저 설정
optimizer = AdamW(model.parameters(), lr=2e-5)

# 훈련 루프 시작
model.train()
for epoch in range(4):  # 에포크 수 필요에 따라 조정
    total_loss = 0
    model.train()
    for batch in tqdm(train_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # 에포크 별로 평균 손실 계산
    epoch_loss = total_loss / len(train_loader)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss}")

    # 검증 세트를 이용한 모델 평가
    model.eval()
    total_eval_accuracy = 0
    for batch in tqdm(val_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)

        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        labels = batch["labels"]

        # 정확도 계산
        accuracy = (predictions == labels).cpu().numpy().mean() * 100
        total_eval_accuracy += accuracy

    # 에포크 별로 검증 정확도 계산
    avg_val_accuracy = total_eval_accuracy / len(val_loader)
    print(f"Validation Accuracy: {avg_val_accuracy:.2f}%")

    # 모델을 저장할 경로
torch.save(model.state_dict(), '/content/drive/My Drive/코드 유사성 판단/model_graphcodebert4.pth')
```

4. 모델링

3) 모델링 코드- 첫번째 모델 train 202만쌍, epoch수 3

```python
# GraphCodeBERT 모델 및 토크나이저 로드
model_name = "microsoft/graphcodebert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.truncation_side = 'left'
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2).to(device)


# 훈련 데이터와 검증 데이터셋 생성
train_dataset = CodePairDataset(tokenizer, train_df, max_length=512)
val_dataset = CodePairDataset(tokenizer, val_df, max_length=512, include_labels=True)

# 데이터 로더 생성
train_loader = DataLoader(train_dataset, batch_size=48, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=48, shuffle=False)

# 옵티마이저 및 손실 함수 설정
optimizer = AdamW(model.parameters(), lr=2e-5)

# 훈련 루프 시작
model.train()
for epoch in range(3):  # 예: 3 에포크 동안 훈련
    total_loss = 0
    model.train()
    for batch in tqdm(train_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # 에포크 평균 훈련 손실 계산
    epoch_loss = total_loss / len(train_loader)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss}")

    # 검증 데이터로 모델 평가
    model.eval()
    total_eval_accuracy = 0
    for batch in tqdm(val_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)

        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        labels = batch["labels"]

        # 정확도 계산
        accuracy = (predictions == labels).cpu().numpy().mean() * 100
        total_eval_accuracy += accuracy

    # 에포크 평균 검증 정확도 계산
    avg_val_accuracy = total_eval_accuracy / len(val_loader)
    print(f"Validation Accuracy: {avg_val_accuracy:.2f}%")

# 모델을 구글 드라이브에 저장
torch.save(model.state_dict(), '/content/drive/My Drive/코드 유사성 관련/model_graphcodebert6.pth')
```

4. 모델링

3) 모델링 코드 - 최종 모델

- 두 모델에서 구한 확률 평균을 최종 평균으로 정함/ 여기서 확률은 simillar 값이 0 또는 1에 속할 확률

```python
model_path_1 = '/content/drive/My Drive/코드 유사성 판단/model_graphcodebert4.pth'
model_path_2 = '/content/drive/My Drive/코드 유사성 판단/model_graphcodebert6.pth'

# 모델 및 토크나이저 초기화
model_name = "microsoft/graphcodebert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.truncation_side = 'left'

# 테스트 데이터셋 준비
test_dataset = CodePairDataset(tokenizer, test_df, max_length=512, include_labels=False)
test_loader = DataLoader(test_dataset, batch_size=48, shuffle=False)

def evaluate_model(model_path):
    model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2).to(device)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()

    probabilities = []
    with torch.no_grad():
        for batch in test_loader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            probs = torch.softmax(logits, dim=-1).cpu().numpy()  # 확률로 변환
            probabilities.extend(probs)

    return np.array(probabilities)

# 각 모델을 평가하여 확률을 얻음
probabilities_1 = evaluate_model(model_path_1)
probabilities_2 = evaluate_model(model_path_2)

# 확률 평균을 최종 모델의 확률로 설정
final_probabilities = (probabilities_1 + probabilities_2) / 2
final_predictions = np.argmax(final_probabilities, axis=1)

# 최종 예측 결과를 sample_submission.csv에 저장
sample_submission['similar'] = final_predictions
sample_submission.to_csv('/content/drive/My Drive/코드 유사성 판단/final_submission.csv', index=False)

# 제출 파일 다운로드
from google.colab import files
files.download('/content/drive/My Drive/코드 유사성 판단/final_submission.csv')
```

결과

- Accuracy (public 점수 기준)

1. 첫번째 모델: 0.9838449666
2. 두번째 모델: 0.9828297033
3. 최종 모델: 0.9858309038

감사합니다