DACON - 코드 유사성 판단 시즌2 AI 경진대회

[Private 8위/0.9824] neulab/codebert-cpp Ensemble(Soft-Voting)

구파일방 팀
김대건, 김립, 정진명

INDEX

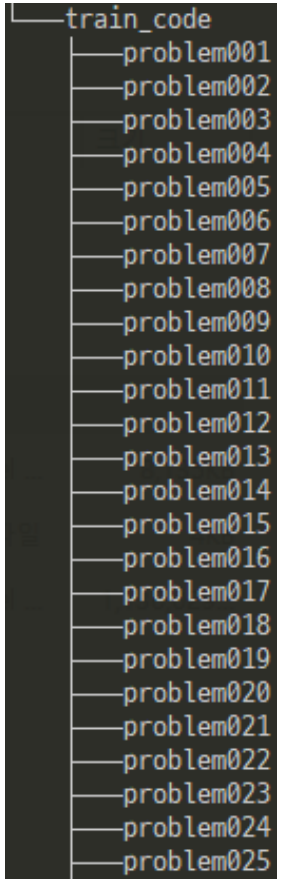
1. CPP 코드 분석

Data Analysis

- 500개의 문제 카테고리, 각 문제 당 500 개의 C++ 코드
  - 동일한 문제 카테고리에 속한 코드는 동일한 작업을 수행

- 데이터 셋의 부족
  - 문제 카테고리를 기준으로 코드 유사성에 대한 labeling 작업 수행
    - 참고 코드 : 코드 유사성 판단 시즌1 – 청소 님
    - 결과 : 1억 212만 5천 건 생성

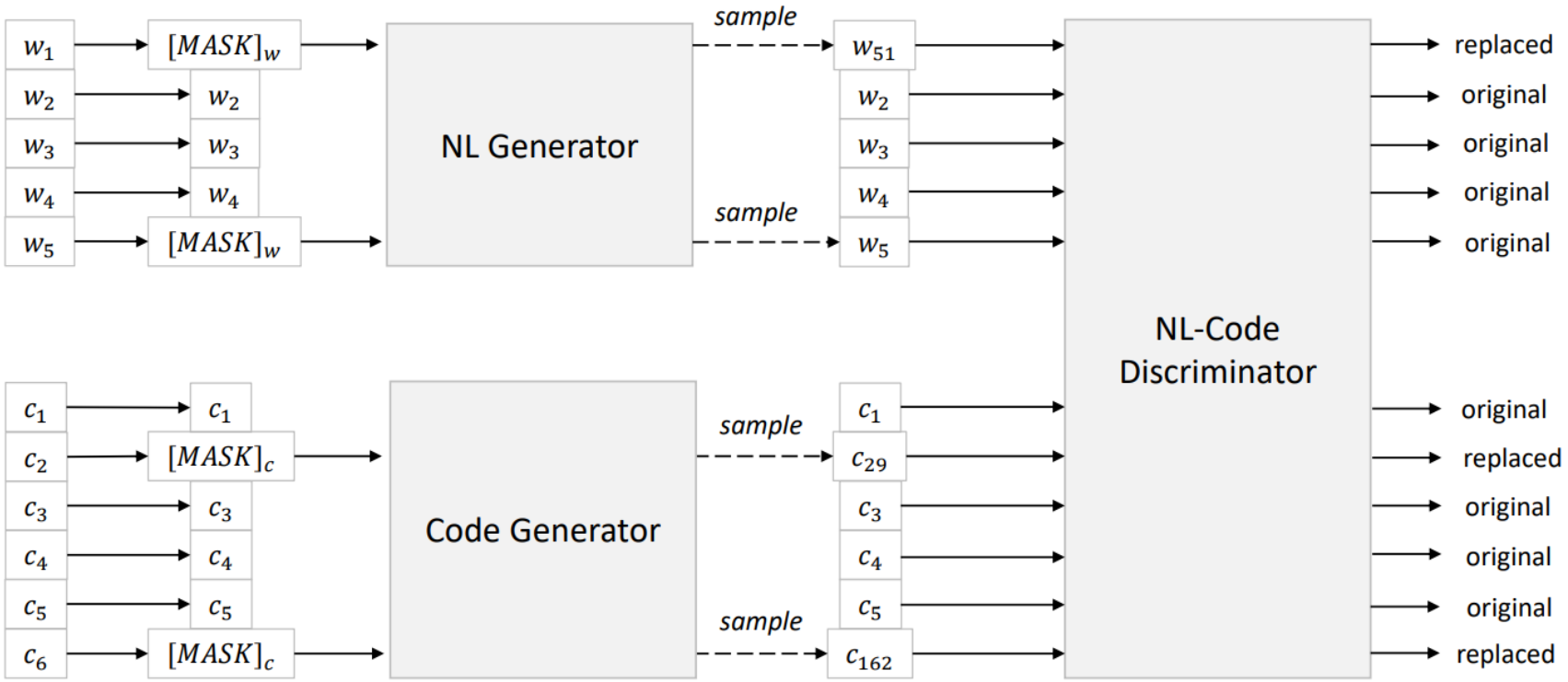
- C++ 전처리 필요
  - 짧은 주석, 긴 주석 삭제



2. C++ 사전학습 언어모델

Architecture(codebert-cpp)

- microsoft/codebert-base-mlm에 codeparrot/github-code-clean 데이터를 학습한 오픈소스 언어모델
- Natural Language와 Programming Language의 의미적 유사성을 비교 학습한 CodeBERT를 사용하였음
- Generator가 token 단위로 replace를 발생시키고 이를 판별하는 Discriminator가 범용적인 표현을 학습함
- Fine-tuning 단계에서 Generator는 제거됨



3. 실험 방법

Hyperparameter

- epoch=1
- learning_rate=2e-5
- batch_size=32
- max_length=512
- loss=CE
- optimizer=AdamW

Environment(RunPod)

- os: ubuntu22.04
- cpu: 16v AMD
- gpu: rtx 4090
- vram: 62GB

Library

- torch==2.1.0+cu118
- transformers==4.39.2
- pandas==2.2.1
- scikit-learn==1.4.1.post1
- 자세한 버전은 requirements.txt 참고 부탁드립니다!

3. 실험 방법

실험 과정

- 전처리
  - 긴 주석, 짧은 주석, 빈 줄 제거

```python
# 전처리 함수 1_주석 처리
def clean_data(text):
    text = text.strip()
    text = re.sub(r"//.*", "", text)
    text = re.sub(r'/\*.*?\*/', '', text, flags=re.DOTALL)
    text = text.strip()
    return text

# 전처리 함수 2_빈 줄 제거
def get_rid_of_empty(c):
    ret = []
    splitted = c.split('\n')
    for s in splitted:
        if len(s.strip()) > 0:
            ret.append(s)
    return '\n'.join(ret)
```

```python
# 전처리 실행
all_code_list_clean = []

for i in range(500):
    cleans = []
    for j in range(500):
        clean = get_rid_of_empty(clean_data(all_code_list[i][j]))
        cleans.append(clean)
    all_code_list_clean.append(cleans)
```

3. 실험 방법

실험 과정

- 모델
  - C++ 사전학습 언어모델 사용(https://huggingface.co/neulab/codebert-cpp)

```
class config():
    def __init__(self):

        self.source_len=512
        self.epochs = 1
        self.learning_rate=2e-5
        self.batch_size=32
        self.shuffle = True
        self.seed=2022
        self.num_labels=2
        self.checkpoint_path = 'neulab/codebert-cpp'
        self.train_path1 = './data/train1.pkl'
        # self.train_path2 = './data/train2.pkl'
        # self.train_path3 = './data/train3.pkl'
        # self.train_path4 = './data/train4.pkl'
        # self.train_path5 = './data/train5.pkl'
        # self.train_path6 = './data/train6.pkl'
        # self.train_path7 = './data/train7.pkl'
        # self.train_path8 = './data/train8.pkl'

cfg = config()
```

```
# 허깅페이스에서 사전학습된 모델 불러옵니다
model = AutoModelForSequenceClassification.from_pretrained(cfg.checkpoint_path, num_labels=cfg.num_labels, output_hidden_states=False).to(device)
tokenizer = AutoTokenizer.from_pretrained(cfg.checkpoint_path,)

tokenizer.truncation_side = "left"
model.resize_token_embeddings(len(tokenizer))
'''DataParallel이 필요할 경우에는 아래 코드를 실행해야 합니다'''
# model = nn.DataParallel(model).to(device)
```

3. 실험 방법

실험 과정

- 데이터
  - 입력 : text-Pair, 출력 : 0 or 1
  - 약 1억 건 생성 후 겹치지 않는 500만 건의 데이터 총 8개(8개 선택 이유는 자원 부족 이슈입니다!)

```python
class CustomDataset(Dataset):

    def __init__(self, data_a,data_b, labels, tokenizer, source_len) :
    # 내가 필요한 것들을 가져와서 선처리
        self.data_a = data_a.copy()
        self.data_b = data_b.copy()
        self.labels = labels.copy()
        self.tokenizer = tokenizer
        self.source_len = source_len

    def __getitem__(self, index) :
    # 데이터 셋에서 한 개의 데이터를 가져오는 함수 정의

        text1 = self.data_a[index]
        text2 = self.data_b[index]

        '''text_pair에 비교할 문장을 입력하면 알아서 sep토큰으로 문장 구분된 하나의 입력 셋이 형성됩니다.'''
        inputs = self.tokenizer(text = text1,text_pair=text2,max_length=self.source_len,padding='max_length',truncation=True, return_tensors='pt')
        label = self.labels[index]

        '''neulab/codebert-cpp는 input_ids와 attention_mask를 입력 받습니다'''
        input_ids = inputs['input_ids'].squeeze()
        attention_mask = inputs['attention_mask'].squeeze()

        inputs_dict = {
            'input_ids' : input_ids.to(device, dtype = torch.long),
            'attention_mask' : attention_mask.to(device, dtype = torch.long),
        }
        label = torch.tensor(label).to(device, dtype = torch.long)


        return inputs_dict, label #

    def __len__(self) :
    # 데이터 셋의 길이
        return len(self.labels)
```

```python
traindf = pd.read_pickle(cfg.train_path1)
# traindf = pd.read_pickle(cfg.train_path2)
# traindf = pd.read_pickle(cfg.train_path3)
# traindf = pd.read_pickle(cfg.train_path4)
# traindf = pd.read_pickle(cfg.train_path5)
# traindf = pd.read_pickle(cfg.train_path6)
# traindf = pd.read_pickle(cfg.train_path7)
# traindf = pd.read_pickle(cfg.train_path8)
```

3. 실험 방법

실험 과정

- 파인 튜닝
  - classification, 1개의 모델은 DataParallel활용

```
for epoch in tqdm(range(1,cfg.epochs+1)):
    t0 = time.time()
    train(epoch, model, optimizer, train_loader)
    torch.save(model.state_dict(), './data/trained_cppbert1.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert2.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert3.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert4.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert5.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert6.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert7.pt')
    # torch.save(model.state_dict(), './data/trained_cppbert8.pt')

    '''DataParallel후 저장하는 방법이 약간 다릅니다'''
    # torch.save(model.module.state_dict(), './data/trained_cppbert1.pt')

    scheduler.step()
```

3. 실험 방법

실험 과정

- 앙상블
  - Soft Voting

```
# model.cuda()

test_tensor = TensorDataset(test_input_ids, test_attention_masks)
test_sampler = SequentialSampler(test_tensor)
test_dataloader = DataLoader(test_tensor, sampler=test_sampler, batch_size=16)

submission = pd.read_csv("./data/sample_submission.csv")

logits_list = [] #soft voting을 위한 리스트입니다
preds = np.array([]) #최종 출력값(레이블)입니다


for step, batch in tqdm(enumerate(test_dataloader), desc="Iteration", smoothing=0.05):
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask = batch

    with torch.no_grad():
        outputs = model(b_input_ids, attention_mask=b_input_mask)

    '''soft voting을 위한 logit값'''
    logits = outputs[0]
    logits = logits.detach().cpu()
    logits_list.append(logits)

    '''최종 출력label(0 & 1)'''
    _pred = logits.numpy()
    pred = np.argmax(_pred, axis=1).flatten()
    preds = np.append(preds, pred)

submission['similar'] = preds
all_logits = torch.cat(logits_list, dim=0)
```

```
import torch
import numpy as np
import pandas as pd

submission = pd.read_csv("./data/sample_submission.csv")

logits_1 = torch.nn.functional.softmax(torch.load("./data/all_logits_model1.pt"))
logits_2 = torch.nn.functional.softmax(torch.load("./data/all_logits_model2.pt"))
logits_3 = torch.nn.functional.softmax(torch.load("./data/all_logits_model3.pt"))
logits_4 = torch.nn.functional.softmax(torch.load("./data/all_logits_model4.pt"))
logits_5 = torch.nn.functional.softmax(torch.load("./data/all_logits_model5.pt"))
logits_6 = torch.nn.functional.softmax(torch.load("./data/all_logits_model6.pt"))
logits_7 = torch.nn.functional.softmax(torch.load("./data/all_logits_model7.pt"))
logits_8 = torch.nn.functional.softmax(torch.load("./data/all_logits_model8.pt"))


logits = (logits_1 + logits_2 + logits_3 + logits_4 + logits_5 + logits_6 + logits_7 + logits_8) / 8
logits_np = logits.numpy()
pred = np.argmax(logits_np, axis=1).flatten()


submission['similar'] = pred
submission.to_csv('./data/final_soft_pred.csv', index=False)
```

4. 실험 결론

실험 인사이트

- 모델 : 다양한 코드 스크립트 사전학습 언어모델 사용 결과, 기대한 성능에 미치지 못함.
  - philomath-1209/programming-language-identification, CodeT5-KDE, microsoft/codebert-base, microsoft/codebert-base-mlm
  - Microsoft 제공 모델들은 CodeSearchNet이라는 데이터로 사전학습 되었는데, 해당 데이터는 Python, Javascript, Ruby, Go, Java, and PHP로 구성되어 있고 C++이 없는 것으로 파악. 따라서, neulab/codebert-cpp로 최종 학습
- 데이터 : 200만 건 학습 후 성능 평가 결과 미세하게 500만 건 학습 모델 보다 떨어짐. 따라서, 500만 건의 8개 데이터 활용
  - 추가로, 500만 건 학습된 모델을 다시 continual 하게 500만 건 학습시킬 때 오히려 성능이 떨어짐. Continual learning을 위한 별다른 장치 없이 학습시켜 발생한 현상 같음. 이에 따라 한 모델은 한 번만 파인 튜닝.
- 앙상블 : 같은 조합에서 Hard Voting의 성능보다 Soft Voting이 약 0.2%p(98% -> 98.24%) 우세. 따라서, Soft Voting 채택
- 파인 튜닝 : 1epoch을 넘어가면 같은 데이터가 들어왔을 때 성능이 저하되는 현상 발견(과적합 의심). 따라서, 각 모델은 1에폭만 학습

4. 실험 결론

실험 결론

- 겹치지 않는 코드들로 구성된 500만 건 데이터를 학습한 8개 모델 Soft Voting 결과 0.9824 달성
- Soft Voting이 Hard Voting 보다 성능이 우세한 경향
- 모델의 차이보다 데이터의 품질에 따라 성능 차이에 큰 영향을 주는 것으로 파악

DACON - 코드 유사성 판단 시즌2 AI 경진대회

[Private 8위/0.9824] neulab/codebert-cpp Ensemble(Soft-Voting)

구파일방 팀
김대건, 김립, 정진명