基于 Bert 实现临床试验筛选标准短文本分类（CHIP-CTC）

马英梓

2022 年 5 月 29 日

1 前言

本次实验基于中文医疗信息处理挑战榜 CBLUE(Chinese Biomedical Language Understanding Evaluation)，根据数据集 CHIP-CTC(CHiP - Clinical Trial Criterion dataset) 完成临床试验筛选标准短文本分类别任务，数据量为：训练集数据 22962 条，验证集数据 7682 条，测试集数据 10000 条。。在 baseline 模型的选择中，我选择使用 BERT-wwm-ext-base 模型 [1]，这是一个由哈工大讯飞联合实验室推出的基于全词掩码（Whole Word Masking）技术的中文预训练模型，配置为 12-layer, 768-hidden, 12-heads, 110M parameters。实验的目的旨在验证 NLP 下游任务训练中的一些 tricks 在该任务中是否 work。

2 任务

2.1 任务介绍

给定事先定义好的 44 种筛选标准语义类别和一系列中文临床试验筛选标准的描述句子，参赛者需返回每一条筛选标准的具体类别。

ID	输入 (筛选标准)	输出 (类别)
迁移	Text(input)	Label(output)
探究	年龄 >80 岁	Age
融合	近期颅内或椎管内手术史	Therapy or Surgery
展示	血糖 <2.7mmol/L	Laboratory Examinations

表 1: 标注数据示例

查看数据集 label 分布，根据图 1 可以明显看出数据集中存在样本类别分布不均匀的现象，因此想要取得一个不错的结果需要首先解决样本类别分布不均匀的问题。因此，我打算对不同 label 的样本做不同数量的数据增强，尽可能平衡样本数量。

2.2 评测指标

本任务的评价指标使用宏观 F1 值 (Macro-F1，或称 Average-F1)。该指标先计算每一类的准确率和召回率，然后对所有类别的准确率和召回率求均值，最后后应用 F1 计算公式。因为对各类别的 Precision 和 Recall 求了平均，所以并没有考虑到数据数量的问题。在这种情况下，Precision 和 Recall 较高的类别对 F1 的影响会较大。

计算公式简化为：

$$Precision_{macro} = \frac{\sum_{i=1}^{n} Precision_i}{n}$$

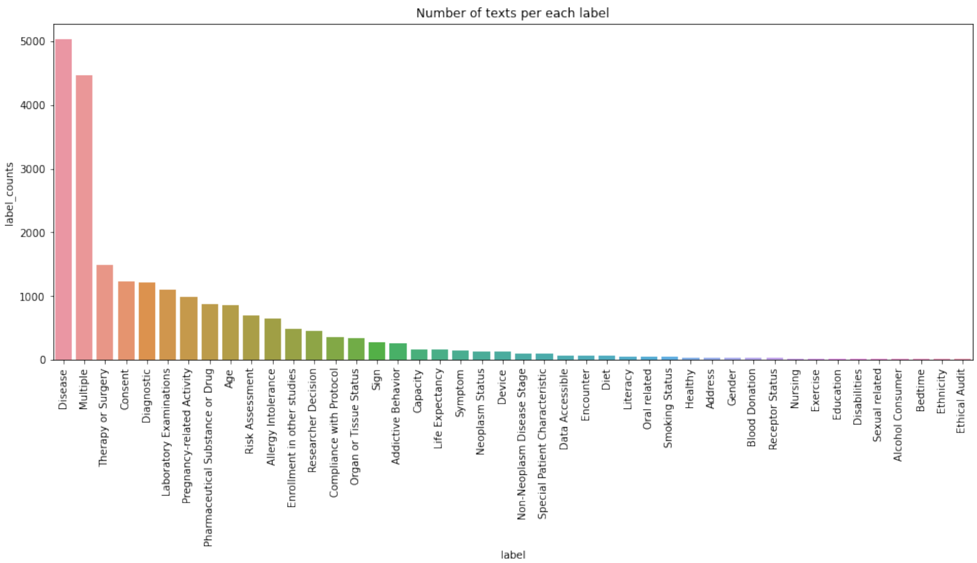

图 1: 数据集中 label 的数量分布

$$Recall_{macro} = \frac{\sum_{i=1}^{n} Recall_i}{n}$$

$$Macro - F1 = 2 \times \frac{Precision_{macro} * Recall_{macro}}{Precision_{macro} + Recall_{macro}}$$

2.3 Tricks

2.3.1 Data Preprocessing

检索数据时发现，每条数据前都有类似（1）、a）这种标号，而这些标号对于文本的分类是没有影响的，因此利用正则表达式去掉。

```
a = re.findall('[\u4e00-\u9fa5A-Za-z][\S\s]+',s,re.S)   #只要字符串中的中文，字母，数字
a = "".join(a)

output:
（4)初诊及复发患者且6个月内未经放，化疗诊治。
初诊及复发患者且6个月内未经放，化疗诊治。
```

2.3.2 Chinese Data Augmentation

这里我使用了 nlpcda 库来对训练文本做数据增强。我从中选择了 3 种方法对文本进行处理。

- 随机同义词替换:

```
from nlpcda import Similarword
smw = Similarword(create_num=3, change_rate=0.3)
rs1 = smw.replace(a)

output:
已知过敏体质的患者， 如对人血白蛋白过敏者,对手术过所需任何药品试剂过敏者;
随机同义词替换>>>>>>
已知过敏体质的病家， 如对人血白蛋白过敏者,对手术过所需另药品试剂过敏者;
已知过敏体质的患者， 如对人血白蛋白过敏者,对手术过所需另药味试剂过敏者;
```


- 随机字删除，同时也能去除尾部的一些标点噪声：

```
from nlpcda import RandomDeleteChar
smw = RandomDeleteChar(create_num=3, change_rate=0.3)
rs1 = smw.replace(a)

output:
有服用镇静剂、抗失眠药物及抗抑郁药的病史，；
随机字删除>>>>>>
服用镇静剂、抗失眠药物抗抑郁药的病史，
有服用镇静剂失眠药物及抗抑郁药的病史
```

- 翻译互转实现的增强，利用百度翻译来回翻译清洗数据：

```
from nlpcda import baidu_translate
en_s = baidu_translate(content=a, appid='20220528001232402', secretKey='6yH2KBcds7tNR1RbynpJ',t_from='zh', t_to='en')
zh_s = baidu_translate(content=en_s, appid='20220528001232402', secretKey='6yH2KBcds7tNR1RbynpJ',t_from='en', t_to='zh')

output:
已知过敏体质的患者，如对人血白蛋白过敏者,对手术过所需任何药品试剂过敏者；
翻译互转>>>>>>
Patients with known allergic constitution, such as those who are allergic to human albumin and those
who are allergic to any drugs and reagents required for surgery;
具有已知过敏体质的患者，如对人类白蛋白过敏的患者和对手术所需的任何药物和试剂过敏的患者；
```

在使用了 3 种数据增强后，我尽可能调整了样本的 label 分布，调整后的结果如图 2 所示。

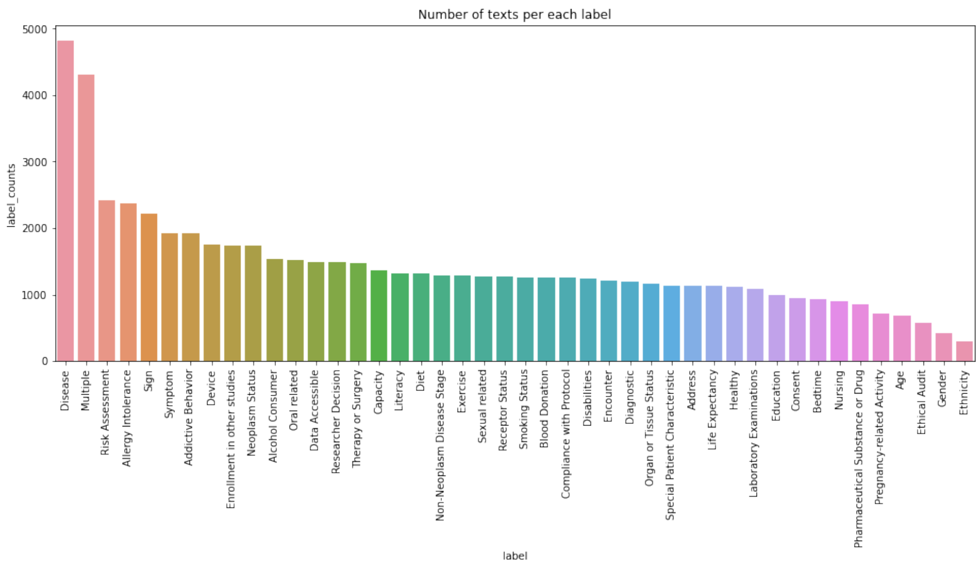

图 2: 数据增强后 label 的数量分布

2.3.3 Adapt Language Models to Domains and Tasks [2]

现在的语言模型（Language Model, LM）大多是在大量且广泛的文本数据上训练而成的，表现优异。该论文作者思考还有没有必要将模型调整迁移到特定目标任务的领域上（domain of a target task），即将已经在大

量且广泛文本上预训练过的模型进行第二阶段的预训练，实验表明，不要停止预训练，对于特定的任务，完全可以用任务相关的数据再对语言模型做二次预训练，能大大提高模型性能。

于是，我借鉴该篇论文的思想，将 CTC 任务的所有数据集一起用于 bert-wwm-ext 的 WWM 预训练任务，再根据二次预训练后的模型在文本分类任务上做 fine-tune，以此验证调整预训练模型的目标域这个方法在该任务上是否有效。

训练超参数设置参考文献 [2]，如表 2 所示：

Computing Infrastructure	RTX 3080ti
Model implementations	https://github.com/gray311/Bert4textclassification4pl

Hyperparameter	Assignment
number of steps	100 epochs (TAPT)
batch size	256(through gradient accumulation)
maximum learning rate	1e-4
learning rate optimizer	AdamW
Adam beta weights	(0.9,0.98)
Weight decay	0.01
learning rate decay	Linear

表 2: Hyperparameters for domain- and task- adaptive pretraining

使用 CrossEntropy 作为损失函数调整 WWM 预训练任务，训练过程如图 3 所示。

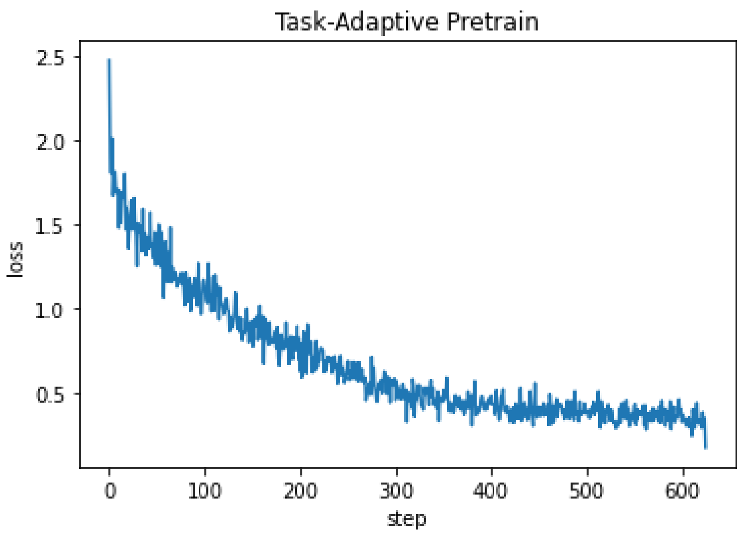

图 3: TAPT 训练过程

2.4 Features from different layers [3]

因为 bert 的不同 layer 对于不同下游任务具有不同的适应性 [3]，比如 k 层对 A 任务效果更好，L 层对 B 任务效果更好，所以做法就是尝试用不同层的输出直接完成下游任务，不一定使用 last_hiddens_layer 作为编码向量。因此，我也尝试了使用 bert-wwm-ext 模型中的不同层输出的特征 token 作为下游任务输出。由于训练

时耗的限制，加上论文中模型错误率表现是随着 layer 层数加深而递减的，因此我只选择了 6-layer,10-layer(0-11layers) 作为特征输出。

3 结果

3.1 Hyperparameters for fine-tune

Hyperparameter	Assignment
number of steps	5 epochs
batch size	16
maximum learning rate	2.5e-5
learning rate optimizer	AdamW
Adam beta weights	(0.9,0.999)
Weight decay	None
learning rate decay	None

表 3: Hyperparameters for fine-tune

3.2 benchmark of tricks

本次实验中，每次做 fine-tune 都随机初始 seed，重复 5 次，求得各个指标的平均值，结果如表 3 所示.

Fine-Tune	Dev.			Test		
	Precision	Recall	Macro-F1	Precision	Recall	Macro-F1
baseline	0.7829	0.8543	0.8060	72.385	65.026	66.661
baseline+process	0.7718	0.8712	0.8092	72.066	66.358	67.241
baseline+process+aug	0.7892	0.8523	0.8136	72.339	65.873	67.411
baseline+process+TAPT	**0.8135**	0.8525	0.8274	72.156	67.412	68.297
baseline+process+aug+TAPT(6layer)	0.7866	0.8773	0.8230	**73.615**	64.584	67.285
baseline+process+aug+TAPT(10layer)	0.7918	0.8714	0.8254	70.903	**68.100**	68.136
baseline+process+aug+TAPT(11layer)	0.8006	**0.8864**	**0.8283**	73.250	66.389	**68.425**

表 4: performance with different tricks on CHIP-CTC

将结果上传至天池 CBLUE2.0 榜单进行评测，结果截图如图 4 所示.

1	2022-05-29 21:55:23	baseline+process+TAPT	已完成	4.878	0.0	68.297	72.156	67.412	编辑
2	2022-05-29 21:51:00	baseline+process+aug	已完成	4.815	0.0	67.411	72.339	65.873	编辑
3	2022-05-29 21:47:24	baseline+process	已完成	4.803	0.0	67.241	72.066	66.358	编辑
4	2022-05-29 21:45:02	baseline	已完成	4.761	0.0	66.661	72.385	65.026	编辑

2022-05-29 22:15:54	baseline+process+aug+TAPT(11layer)	已完成	4.887	68.425	73.250	66.389
2022-05-29 22:14:29	baseline+process+aug+TAPT(6layer)	已完成	4.806	67.285	73.615	64.584
2022-05-29 22:05:16	baseline+process+aug+TAPT(10layer)	已完成	4.867	68.136	70.903	68.100

图 4: 天池 CBLUE2.0 榜单评测结果

对比测试结果可以发现，各项 trick 对模型 Macro-F1 指标都有一定提升效果。在本任务中，bert 模型中最后一层提取的特征效果最好。同时，考虑到 Recall 指标和 Precision 指标往往不能同时达到最优，因此可以考虑一些知识蒸馏，或者集成模型的方法对模型效果进一步提升。另外，仅靠数据增强生成的 samples 有很大相似性，所以不能足够有效的解决 lebel 不平衡的问题，因此可以考虑用一些 few-sample BERT fine-tuning [4] 方法来提高模型精度。

参考文献

[1] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, and Ziqing Yang. Pre-training with whole word masking for chinese bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3504–3514, 2021.

[2] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics.

[3] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019.

[4] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. Revisiting few-sample BERT fine-tuning. *CoRR*, abs/2006.05987, 2020.

备注

本文排版基于 LaTeX【Github】【OverLeaf】【SJTU-ShareLaTeX】模板，采用【CC BY-NC-SA 4.0】进行许可。

A Bert4WWM-Pretrain

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import pandas as pd
import torch
from datasets import Dataset
import datasets
import os
import random
import numpy as np


def seedeverything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True ##
    torch.backends.cudnn.benchmark = True


seedeverything(seed=233)

traindata = pd.read_json('./CHIP-CTC/CHIP-CTC_train.json')
valdata = pd.read_json('./CHIP-CTC/CHIP-CTC_dev.json')
testdata = pd.read_json('./CHIP-CTC/CHIP-CTC_test.json')

examplepreddata = pd.read_excel('./CHIP-CTC/category.xlsx')

examplepreddata['label2idx'] = range(examplepreddata.shape[0])

label2idx = dict(
    zip(examplepreddata['Label Name'], examplepreddata['label2idx']))
idx2label = dict(
    zip(examplepreddata['label2idx'], examplepreddata['Label Name']))

print(idx2label)

traindata['labels'] = [label2idx[item] for item in traindata['label']]
valdata['labels'] = [label2idx[item] for item in valdata['label']]

print(len(traindata))
print(len(valdata))
print(len(testdata))

traindataset = Dataset.from_pandas(traindata)
valdataset = Dataset.from_pandas(valdata)
testdataset = Dataset.from_pandas(testdata)

```



```python
dataset = datasets.DatasetDict({
    'train': traindataset,
    'validation': valdataset,
    'test': testdataset
})

print(dataset)

train_dataset = dataset['train']
val_dataset = dataset['validation']
test_dataset = dataset['test']

print(train_dataset.features)

print(train_dataset[0])

# In[2]:

import argparse
import json
from typing import List

from ltp import LTP
from transformers.models.bert.tokenization_bert import BertTokenizer


def _is_chinese_char(cp):
    """Checks whether CP is the codepoint of a CJK character."""
    # This defines a "chinese character" as anything in the CJK Unicode block:
    #  https://en.wikipedia.org/wiki/CJK_Unified_Ideographs_(Unicode_block)
    #
    # Note that the CJK Unicode block is NOT all Japanese and Korean characters,
    # despite its name. The modern Korean Hangul alphabet is a different block,
    # as is Japanese Hiragana and Katakana. Those alphabets are used to write
    # space-separated words, so they are not treated specially and handled
    # like the all of the other languages.
    if ((cp >= 0x4E00 and cp <= 0x9FFF) or (cp >= 0x3400 and cp <= 0x4DBF) #
            or (cp >= 0x20000 and cp <= 0x2A6DF) #
            or (cp >= 0x2A700 and cp <= 0x2B73F) #
            or (cp >= 0x2B740 and cp <= 0x2B81F) #
            or (cp >= 0x2B820 and cp <= 0x2CEAF) #
            or (cp >= 0xF900 and cp <= 0xFAFF)
            or (cp >= 0x2F800 and cp <= 0x2FA1F) #
        ): #
        return True

    return False


def is_chinese(word: str):
    # word like '180' or '身高' or '神'
    for char in word:
        char = ord(char)
        if not _is_chinese_char(char):
            return 0
    return 1


def get_chinese_word(tokens: List[str]):
    word_set = set()

    for token in tokens:
```

```python
        chinese_word = len(token) > 1 and is_chinese(token)
        if chinese_word:
            word_set.add(token)
    word_list = list(word_set)
    return word_list


def add_sub_symbol(bert_tokens: List[str], chinese_word_set: set()):
    if not chinese_word_set:
        return bert_tokens
    max_word_len = max([len(w) for w in chinese_word_set])

    bert_word = bert_tokens
    start, end = 0, len(bert_word)
    while start < end:
        single_word = True
        if is_chinese(bert_word[start]):
            l = min(end - start, max_word_len)
            for i in range(l, 1, -1):
                whole_word = "".join(bert_word[start:start + i])
                if whole_word in chinese_word_set:
                    for j in range(start + 1, start + i):
                        bert_word[j] = "##" + bert_word[j]
                    start = start + i
                    single_word = False
                    break
        if single_word:
            start += 1
    return bert_word


def prepare_ref(lines: List[str], ltp_tokenizer: LTP,
                bert_tokenizer: BertTokenizer):
    ltp_res = []

    for i in range(0, len(lines), 100):
        res = ltp_tokenizer.seg(lines[i:i + 100])[0]
        res = [get_chinese_word(r) for r in res]
        ltp_res.extend(res)
    assert len(ltp_res) == len(lines)

    bert_res = []
    for i in range(0, len(lines), 100):
        res = bert_tokenizer(lines[i:i + 100],
                             add_special_tokens=True,
                             truncation=True,
                             max_length=512)
        bert_res.extend(res["input_ids"])
    assert len(bert_res) == len(lines)

    ref_ids = []
    for input_ids, chinese_word in zip(bert_res, ltp_res):

        input_tokens = []
        for id in input_ids:
            token = bert_tokenizer._convert_id_to_token(id)
            input_tokens.append(token)
        input_tokens = add_sub_symbol(input_tokens, chinese_word)
        ref_id = []
        # We only save pos of chinese subwords start with ##, which mean is part of a whole word.
        for i, token in enumerate(input_tokens):
            if token[:2] == "##":
```



```
            clean_token = token[2:]
            # save chinese tokens' pos
            if len(clean_token) == 1 and _is_chinese_char(
                    ord(clean_token)):
                ref_id.append(i)
        ref_ids.append(ref_id)

    assert len(ref_ids) == len(bert_res)

    return ref_ids


# In[3]:

from transformers.data.data_collator import DataCollatorForLanguageModeling, DataCollatorForWholeWordMask
from transformers import BertForMaskedLM

path = "hflchinese-bert-wwm-ext"
tokenizer = BertTokenizer.from_pretrained(path)
net = BertForMaskedLM.from_pretrained(path)

from ltp import LTP

ltp = LTP()

sent = [item['text'] for item in train_dataset] + [
    item['text'] for item in val_dataset
] + [item['text'] for item in test_dataset]

ref = prepare_ref(sent, ltp, tokenizer)
print(len(ref))

# In[4]:

from transformers import BertTokenizer, BertModel, AutoModelForSequenceClassification
from datasets import Dataset

tokenizer = BertTokenizer.from_pretrained("hflchinese-bert-wwm-ext")


def tokenize_function(sample):
    return tokenizer(sample['text'], truncation=True)


tokenized_datasets = dataset.map(tokenize_function, batched=True)

tokenized_datasets['train'] = tokenized_datasets['train'].remove_columns(
    ['id', 'text', 'label'])
tokenized_datasets['validation'] = tokenized_datasets[
    'validation'].remove_columns(['id', 'text', 'label'])
tokenized_datasets['test'] = tokenized_datasets['test'].remove_columns(
    ['id', 'text'])

# In[5]:

encoder_dict = tokenized_datasets['train']['input_ids'] + tokenized_datasets[
    'validation']['input_ids'] + tokenized_datasets['test']['input_ids']

# 加上子字信息，而且传入的是List，不是tensor。
train_mlm_dataset = [{
    'input_ids': encoder_dict[i],
    'chinese_ref': ref[i]
```

```
} for i in range(len(ref))]

datacollecter = DataCollatorForWholeWordMask(tokenizer)

# In[6]:

from torch.utils.data import DataLoader
from transformers import DataCollatorWithPadding #实现按batch自动padding

train_mlm_dataloader = DataLoader(train_mlm_dataset,
                          shuffle=True,
                          batch_size=8,
                          collate_fn=datacollecter)
for batch in train_mlm_dataloader:
   print({k: v.shape for k, v in batch.items()})
   break

# In[7]:

for batch in train_mlm_dataloader:
   outputs = net(**batch)
   print(outputs)
   break

# In[8]:

import pytorch_lightning as pl
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import EarlyStopping
from pytorch_lightning.callbacks import ModelCheckpoint
from tensorboardX import SummaryWriter
from transformers import AdamW, get_scheduler

from datasets import load_metric
from statistics import mean
from sklearn import metrics
from torch import nn
import json
import warnings

warnings.filterwarnings("ignore")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_epochs = 30
lr = 2e-5

num_training_steps = num_epochs * len(
   train_mlm_dataloader) # num of batches * num of epochs
print(num_training_steps)


class Bert4wwmtask_lightningsystem(pl.LightningModule):

   def __init__(self, net, lr, epoch, len):
      super(Bert4wwmtask_lightningsystem, self).__init__()
      self.net = net.to(device)
      self.lr = lr
      self.epoch = epoch
      self.num_training_steps = len
      self.writer = SummaryWriter('./log')
      self.iteration = 0
      self.num = 0
```



```
        #self.metric = load_metric("glue", "mrpc",mirror="tuna")

    def configure_optimizers(self):

        self.optimizer = AdamW(self.net.parameters(),
                               lr=self.lr,
                               betas=(0.9, 0.98),
                               weight_decay=0.01)
        self.lr_scheduler = get_scheduler(
            'linear',
            optimizer=self.optimizer,
            num_warmup_steps=0,
            num_training_steps=self.num_training_steps)
        optim_dict = {
            'optimizer': self.optimizer,
            'lr_scheduler': self.lr_scheduler
        }
        return optim_dict

    def training_step(self, batch, batch_idx):
        batch = {k: v.to(device) for k, v in batch.items()}
        loss = self.net(**batch).loss
        lr_ = self.lr * (1.0 - self.iteration / self.num_training_steps)**0.9
        for param_group in self.optimizer.param_groups:
            param_group['lr'] = lr_
        self.iteration += 1
        self.writer.add_scalar(
            'info/lr',
            self.optimizer.state_dict()['param_groups'][0]['lr'],
            self.iteration)
        self.writer.add_scalar('info/loss', loss, self.iteration)
        return loss

    def validation_step(self, batch, batch_idx):
        pass

    def test_step(self, batch, batch_idx):
        pass

    def training_epoch_end(self, outputs):
        self.num += 1
        if self.num % 10 == 0:
            pt_save_directory = "./pt_save_pretrained" + str(self.num)
            self.net.save_pretrained(pt_save_directory)

    def validation_epoch_end(self, outputs):
        pass


# In[9]:

model = Bert4wwmtask_lightningsystem(net=net,
                                     lr=lr,
                                     epoch=num_epochs,
                                     len=num_training_steps)

trainer = Trainer(
    logger=False,
    max_epochs=num_epochs,
    gpus=1,
    reload_dataloaders_every_n_epochs=False,
    num_sanity_val_steps=0, # Skip Sanity Check
```



```
    #callbacks=[checkpoint_callback],
    #limit_train_batches=0.05
    precision=16,
    accumulate_grad_batches=32,
    #gradient_clip_val=0.5,
)

trainer.fit(model, train_mlm_dataloader)
```

src/bert4wwm.py

B Bert4CTC-Fine-tune

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:

import pandas as pd
import torch
from datasets import Dataset
import datasets
import os
import random
import numpy as np
import re
from copy import deepcopy
import json
import pytorch_lightning as pl
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import EarlyStopping
from pytorch_lightning.callbacks import ModelCheckpoint
from nlpcda import Similarword


def seedeverything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True ##
    torch.backends.cudnn.benchmark = True


seedeverything(seed=233)

traindata = pd.read_json('./CHIP-CTC/CHIP-CTC_train_aug.json')
valdata = pd.read_json('./CHIP-CTC/CHIP-CTC_dev.json')
testdata = pd.read_json('./CHIP-CTC/CHIP-CTC_test.json')
testdata_temp = deepcopy(testdata)


def textclean(x):
    a = re.findall('[\u4e00-\u9fa5A-Za-z][\S\s]+', x, re.S)
    a = "".join(a)
    return a

```



```python
traindata['text'] = traindata['text'].apply(lambda x: textclean(x))
valdata['text'] = valdata['text'].apply(lambda x: textclean(x))
testdata['text'] = testdata['text'].apply(lambda x: textclean(x))

examplepreddata = pd.read_excel('./CHIP-CTC/category.xlsx')

examplepreddata['label2idx'] = range(examplepreddata.shape[0])

label2idx = dict(
   zip(examplepreddata['Label Name'], examplepreddata['label2idx']))
idx2label = dict(
   zip(examplepreddata['label2idx'], examplepreddata['Label Name']))

with open("idx2label.json", "w", encoding="utf-8") as fp:
   json.dump(idx2label, fp, ensure_ascii=False, indent=4)

print(idx2label)

traindata['labels'] = [label2idx[item] for item in traindata['label']]
valdata['labels'] = [label2idx[item] for item in valdata['label']]

print(len(traindata))
print(len(valdata))
print(len(testdata))

traindataset = Dataset.from_pandas(traindata)
valdataset = Dataset.from_pandas(valdata)
testdataset = Dataset.from_pandas(testdata)

dataset = datasets.DatasetDict({
   'train': traindataset,
   'validation': valdataset,
   'test': testdataset
})

print(dataset)

train_dataset = dataset['train']
print(train_dataset.features)

print(train_dataset[0])

# In[4]:

from transformers import BertTokenizer, BertModel, AutoModelForSequenceClassification

checkpoint = "hflchinese-bert-wwm-withpretrain-ext"
tokenizer = BertTokenizer.from_pretrained(checkpoint)


def tokenize_function(sample):
   return tokenizer(sample['text'], truncation=True)


tokenized_datasets = dataset.map(tokenize_function, batched=True)

tokenized_datasets['train'] = tokenized_datasets['train'].remove_columns(
   ['id', 'text', 'label'])
tokenized_datasets['validation'] = tokenized_datasets[
   'validation'].remove_columns(['id', 'text', 'label'])
tokenized_datasets['test'] = tokenized_datasets['test'].remove_columns(
   ['id', 'text'])
```



```python
from transformers import DataCollatorWithPadding #实现按batch自动padding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

print(tokenized_datasets)

# In[6]:

from torch.utils.data import DataLoader, Dataset

train_dataloader = DataLoader(tokenized_datasets['train'],
                              shuffle=True,
                              batch_size=8,
                              collate_fn=data_collator)
val_dataloader = DataLoader(tokenized_datasets['validation'],
                            batch_size=8,
                            collate_fn=data_collator)
test_dataloader = DataLoader(tokenized_datasets['test'],
                             batch_size=8,
                             collate_fn=data_collator)
for batch in test_dataloader:
    print({k: v.shape for k, v in batch.items()})
    break

# In[7]:

from transformers import BertTokenizer, BertModel, AutoModelForSequenceClassification, AutoModel
from transformers import BertForSequenceClassification, BertForMaskedLM

net = AutoModelForSequenceClassification.from_pretrained(
    checkpoint, num_labels=examplepreddata.shape[0])

for batch in train_dataloader:
    outputs = net(**batch)
    print(outputs)
    break

# In[8]:
'''
3.8546

-1.4115e-01

4.1694
'''
from transformers import AdamW, get_scheduler
from datasets import load_metric
from statistics import mean
from sklearn import metrics
from torch import nn
import json
import warnings
from tensorboardX import SummaryWriter

warnings.filterwarnings("ignore")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_epochs = 5
lr = 2.5e-5
num_labels = examplepreddata.shape[0]
num_training_steps = num_epochs * len(
```

```
    train_dataloader) # num of batches * num of epochs
print(num_training_steps)


class Mlp(nn.Module):

    def __init__(self,
                 in_features,
                 hidden_features=1000,
                 out_features=None,
                 act_layer=nn.GELU,
                 drop=0.):
        super().__init__()
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.softmax = nn.Softmax(dim=-1)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return self.softmax(x)


class Bert4textclassification_lightningsystem(pl.LightningModule):

    def __init__(self, net, lr, epoch, len):
        super(Bert4textclassification_lightningsystem, self).__init__()
        self.net = net.to(device)
        self.lr = lr
        self.epoch = epoch
        self.num_training_steps = len
        self.writer = SummaryWriter('./log-' + checkpoint)
        self.iteration = 0
        #self.metric = load_metric("glue", "mrpc",mirror="tuna")

    def configure_optimizers(self):

        self.optimizer = AdamW(self.net.parameters(), lr=self.lr)
        lr_scheduler = get_scheduler(
            'linear',
            optimizer=self.optimizer,
            num_warmup_steps=0,
            num_training_steps=self.num_training_steps)
        optim_dict = {
            'optimizer': self.optimizer,
            'lr_scheduler': lr_scheduler
        }
        return optim_dict

    def metrics_compute(self, mode, outputs):
        loss = []
        loss.append(outputs[0][mode + '_loss'])
        predictions = outputs[0]['predictions']
        labels = outputs[0]['labels']
        for i in range(1, len(outputs)):
            loss.append(outputs[i][mode + '_loss'])
            predictions = torch.concat(
```



```
                [predictions, outputs[i]['predictions']], dim=0)
            labels = torch.concat([labels, outputs[i]['labels']], dim=0)
        loss = torch.tensor(loss)
        predictions = predictions.cpu().detach().numpy()
        labels = labels.cpu().detach().numpy()
        return loss, predictions, labels

    def training_step(self, batch, batch_idx):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = self.net(**batch)
        loss = outputs.loss
        lr_ = self.lr * (1.0 - self.iteration / self.num_training_steps)**0.9
        for param_group in self.optimizer.param_groups:
            param_group['lr'] = lr_
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        metrics_dict = metrics.classification_report(
            predictions.cpu().detach().numpy(),
            batch['labels'].cpu().detach().numpy(),
            digits=4,
            output_dict=True)
        self.writer.add_scalar('info/loss', loss, self.iteration)
        self.writer.add_scalar('info/weighted_avg',
                               metrics_dict['weighted avg']['f1-score'],
                               self.iteration)
        self.iteration += 1
        return loss

    def validation_step(self, batch, batch_idx):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = self.net(**batch)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        metrics_dict = metrics.classification_report(
            predictions.cpu().detach().numpy(),
            batch['labels'].cpu().detach().numpy(),
            digits=4,
            output_dict=True)
        self.log('macro_avg', metrics_dict['macro avg']['f1-score'])
        #self.metric.add_batch(predictions=predictions, references=batch["labels"])
        return {
            'val_loss': outputs.loss,
            'predictions': predictions,
            'labels': batch['labels']
        }

    def test_step(self, batch, batch_idx):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = self.net(**batch)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)
        return {'test_loss': outputs.loss, 'predictions': predictions}

    def training_epoch_end(self, outputs):
        pass

    def validation_epoch_end(self, outputs):
        print(outputs[0]['predictions'].shape)
        print(len(outputs))
        val_loss, predictions, labels = self.metrics_compute('val', outputs)
        print(predictions.shape)
        print('\n', "val_loss: ", val_loss.mean())
```



```
        print(metrics.classification_report(predictions, labels, digits=4))

    def test_epoch_end(self, outputs):
        predictions = outputs[0]['predictions']
        for i in range(1, len(outputs)):
            predictions = torch.concat(
                [predictions, outputs[i]['predictions']], dim=0)
        predictions = predictions.cpu().detach().numpy().tolist()
        test_labels = [idx2label[idx] for idx in predictions]
        testdata_temp['label'] = test_labels
        test_pred_list = []
        for i in range(testdata_temp.shape[0]):
            temp_dict = {}
            temp_dict['id'] = testdata_temp.iloc[i, 0]
            temp_dict['label'] = testdata_temp.iloc[i, 2]
            temp_dict['text'] = testdata_temp.iloc[i, 1]
            test_pred_list.append(temp_dict)
        print('\n', testdata_temp.head())
        with open("result.json", "w", encoding="utf-8") as fp:
            json.dump(test_pred_list, fp, ensure_ascii=False, indent=4)


# In[ ]:

model = Bert4textclassification_lightningsystem(net, lr, num_epochs,
                                                num_training_steps)
checkpoint_callback = ModelCheckpoint(
    monitor='macro_avg',
    dirpath='./output_baseline',
    filename='hflchinese-bert-wwm-ext-CHIP-CTC-{epoch:02d}-{macro_avg:.4f}',
    mode='max',
    save_top_k=-1,
)
trainer = Trainer(
    logger=False,
    max_epochs=num_epochs,
    gpus=1,
    reload_dataloaders_every_n_epochs=False,
    num_sanity_val_steps=0, # Skip Sanity Check
    callbacks=[checkpoint_callback],
    #limit_train_batches=0.05
    #precision=16,
    accumulate_grad_batches=2,
    #gradient_clip_val=0.5,
)

trainer.fit(model, train_dataloader, val_dataloader)

# In[ ]:

print(testdata.shape)
print(testdata_temp.shape)

model = Bert4textclassification_lightningsystem.load_from_checkpoint(
    checkpoint_path=
    './outputhflchinese-bert-wwm-withpretrain-ext/hflchinese-bert-wwm-ext-CHIP-CTC-epoch=03-weighted_avg=0.8530.ckpt',
    net=net,
    lr=lr,
    epoch=num_epochs,
    len=num_training_steps)

trainer = Trainer(
```



```
    logger=False,
    gpus=1,
    #limit_train_batches=0.05
    #precision=16,
    #accumulate_grad_batches=2,
    #gradient_clip_val=0.5,
)
trainer.test(model=model, dataloaders=test_dataloader)
```

src/bert4CTC.py

C Chinese Data Augmentation

```
#!/usr/bin/env python
# coding: utf-8

# In[51]:

import pandas as pd
import torch
from datasets import Dataset
import datasets
import os
import random
import numpy as np
import re
from copy import deepcopy
import json
import pytorch_lightning as pl
from pytorch_lightning import Trainer
from pytorch_lightning.callbacks import EarlyStopping
from pytorch_lightning.callbacks import ModelCheckpoint
from nlpcda import Similarword


def seedeverything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True ##
    torch.backends.cudnn.benchmark = True


seedeverything(seed=233)

traindata = pd.read_json('./CHIP-CTC/CHIP-CTC_train.json')
valdata = pd.read_json('./CHIP-CTC/CHIP-CTC_dev.json')
testdata = pd.read_json('./CHIP-CTC/CHIP-CTC_test.json')
testdata_temp = deepcopy(testdata)

print(traindata.head())

# In[52]:

import matplotlib.pyplot as plt
import seaborn as sns

```



```python
# In[53]:

temp = traindata.groupby(["label"])['text'].nunique()
df = pd.DataFrame({'label': temp.index, 'label_counts': temp.values})
df = df.sort_values(['label_counts'], ascending=False).head(50)
plt.figure(figsize=(16, 6))
plt.title(f'Number of texts per each label')
sns.set_color_codes("pastel")
s = sns.barplot(x='label', y="label_counts", data=df)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
locs, labels = plt.xticks()
plt.show()

# In[54]:

print(df.head())
label2num = dict(zip(df['label'], df['label_counts']))
print(label2num)
print(label2num['Therapy or Surgery'])

# In[57]:


class TextAugmentation:

   def __init__(self, traindata, valdata, testdata, label2num, padding,
            change_rate):
      super(TextAugmentation, self).__init__()
      self.traindata = traindata
      self.valdata = valdata
      self.testdata = testdata
      self.label2num = label2num
      self.padding = padding
      self.change_rate = change_rate

   def textclean(self, x):
      a = re.findall('[\u4e00-\u9fa5A-Za-z][\S\s]+', x, re.S)
      a = "".join(a)
      return a

   def TextProcess(self):
      self.traindata['text'] = self.traindata['text'].apply(
         lambda x: self.textclean(x))
      self.valdata['text'] = self.valdata['text'].apply(
         lambda x: self.textclean(x))
      self.testdata['text'] = self.testdata['text'].apply(
         lambda x: self.textclean(x))

   def TextAug(self):
      self.traindata['res'] = ""
      for i in range(self.traindata.shape[0]):
         if self.padding > self.label2num[self.traindata.iloc[i, 1]]:
            res = max(
               math.ceil(
                  math.sqrt((self.padding -
                          self.label2num[self.traindata.iloc[i, 1]]) /
                         self.label2num[self.traindata.iloc[i, 1]])),
               1)
         else:
            res = 1
         self.traindata.iloc[i, 3] = res

```

```
        for i in tqdm(range(self.traindata.shape[0])):
            a = self.traindata.iloc[i, 2]
            res = self.traindata.iloc[i, 3]
            smw = CharPositionExchange(create_num=res,
                                       change_rate=self.change_rate,
                                       char_gram=3)
            rs1 = smw.replace(a)[1:]
            if len(rs1) > 0:
                temp = self.traindata.iloc[i]
                temp = temp.to_frame()
                temp = pd.DataFrame(temp.values.T, columns=temp.index)
                temp = pd.concat([temp] * len(rs1), ignore_index=True)
                temp['text'] = rs1
                self.traindata = pd.concat([self.traindata, temp],
                                           ignore_index=True)

        for i in tqdm(range(self.traindata.shape[0])):
            a = self.traindata.iloc[i, 2]
            res = self.traindata.iloc[i, 3]
            smw = RandomDeleteChar(create_num=res,
                                   change_rate=self.change_rate)
            rs1 = smw.replace(a)[1:]
            if len(rs1) > 0:
                temp = self.traindata.iloc[i]
                temp = temp.to_frame()
                temp = pd.DataFrame(temp.values.T, columns=temp.index)
                temp = pd.concat([temp] * len(rs1), ignore_index=True)
                temp['text'] = rs1
                self.traindata = pd.concat([self.traindata, temp],
                                           ignore_index=True)

    def get_aug_text(self):
        return self.traindata, self.valdata, self.testdata


# In[58]:

temp_train_data = traindata
df = temp_train_data
temp = df.groupby(["label"])['text'].nunique()
df = pd.DataFrame({'label': temp.index, 'label_counts': temp.values})
df = df.sort_values(['label_counts'], ascending=False).head(50)
plt.figure(figsize=(16, 6))
plt.title(f'Number of texts per each label')
sns.set_color_codes("pastel")
s = sns.barplot(x='label', y="label_counts", data=df)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
locs, labels = plt.xticks()
plt.show()

# In[60]:

label2num = dict(zip(df['label'], df['label_counts']))
text_aug = TextAugmentation(temp_train_data,
                            valdata,
                            testdata,
                            label2num,
                            padding=1500,
                            change_rate=0.3)
text_aug.TextProcess()
text_aug.TextAug()
traindata, valdata, testdata = text_aug.get_aug_text()
```

```python
print(len(traindata))
print(len(valdata))
print(len(testdata))

# In[61]:

df = traindata
print(traindata)
temp = df.groupby(["label"])['text'].nunique()
df = pd.DataFrame({'label': temp.index, 'label_counts': temp.values})
df = df.sort_values(['label_counts'], ascending=False).head(50)
plt.figure(figsize=(16, 6))
plt.title(f'Number of texts per each label')
sns.set_color_codes("pastel")
s = sns.barplot(x='label', y="label_counts", data=df)
s.set_xticklabels(s.get_xticklabels(), rotation=90)
locs, labels = plt.xticks()
plt.show()
```

src/data_augmentation.py

