PPCA CodeMate项目报告

房诗涵

2023 年 7 月 28 日

1 爬虫

1.1 使用request

使用request发起请求，将响应解析为json（CSDN）或取响应的文本用lxml.etree.HTML转化为DOM树，使用xpath定位获取text信息。

1.2 使用selenium

使用selenium的webdriver模拟网站登录，能用获得渲染后的网页页面，ActionChains实现点击等功能，使用driver.execute_script(js)执行JavaScript实现向下滑动等功能。

同时，在使用过程中发现，使用selenium爬取Stackoverflow，不会受到网站人机验证的限制。

1.3 使用并行

1.3.1 协程

essence_link.py使用协程，利用Queue在协程间通信，在 `load_page`做网络请求时，切换协程做解析页面快照任务。

1.3.2 线程

Stackoverflow/get_detail.py使用threading实现多线程，用 `threading.Semaphore(14)`限制最多线程数为14，将访问每一个地址作为一个task被执行。

1.4 使用scrapy

用scrapy框架能用更少的代码实现爬虫操作。

scdnSpider是用scrapy框架实现的CSDN爬虫，主要实现新建spider项目的pipeline.py（对爬取的数据的处理）和spider（爬取数据），然后在setting.py中指定pipeline的组织方式。

在使用过程中，发现spider的域名限制不会将域名外的网址入队，一方面，能筛选掉部分无用网址，但是如果从当前网页获得的目标网址域名发生变化，爬取任务将无法执行。

2 爬虫数据分类器训练

2.1 jieba分词

使用中文分词库jieba，将测试集和训练集的问答对分解为词，并去除停用词。停用词包括中文标点、语气词等。

分词前后示例如下

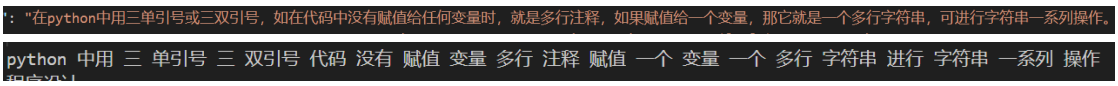


图 1: 中文文本分词前后

在初步分词后，发现存在部分专业词汇被分解为多个词素，如“结构体”等被拆分为“结构”、“体”。尝试使用`jieba.load_userdict('file_name')`函数，指定这些词汇为一个整体。尝试过去除待分词的字符串中的所有标点、空格，发现会导致代码块被作为一长串英文字符串，无法被正确区分。

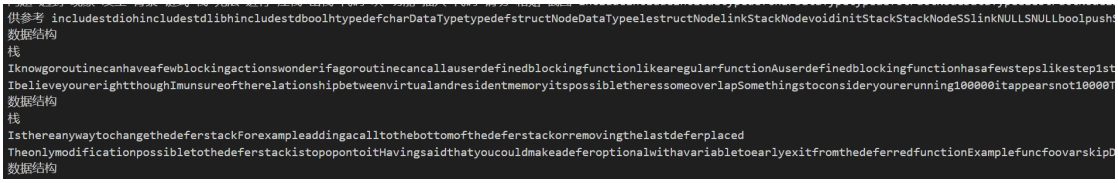


图 2: 忽略字符串标点符号和空格后的分词结果

在jieba分词阶段，如何合理地分出代码块是一个难题。一个想法是通过用代码块的标识符正则匹配区分，如完整的c++代码往往以`#include`开头，这个方法存在的问题是难以确定代码块的结束标记，而且问答对中代码不一定为完整的c++代码，适用范围

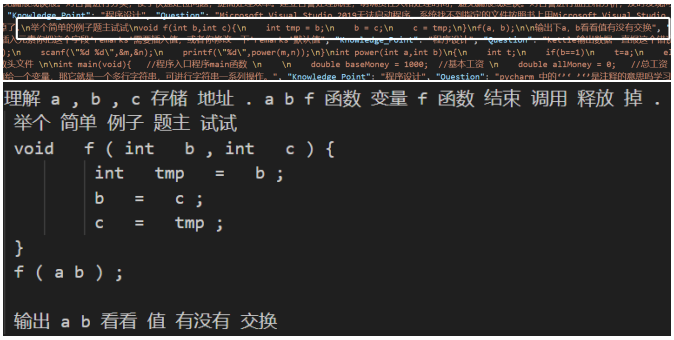


图 3: 未特殊处理的代码块分词结果

小，匹配的成功率低。另一个想法是以“‘作为区分，爬虫爬取的代码段大多用markdown书写，被包在“‘内。在第三周的爬虫任务中，通过在爬取时，在code元素文本前后加上“‘，方便文本中代码块的识别。

由于分类目标是将问答对分辨为“高质量”和“低质量”，我认为是否带有代码块可以作为一个区分问答对质量高低的依据，而代码块具体内容由于差异极大，对分类不具有很大作用，于是决定将所有代码块作为一个统一的“词”。使用正则表达式匹配相邻两个“‘包裹的文本，并使用re库替换为codeBlock。

```
import re

pat= r'‘‘‘(.|\\n)*?‘‘‘'
replacement="codeBlock"
replaced_string = re.sub(pattern=pat,repl= replacement, string=str)
```

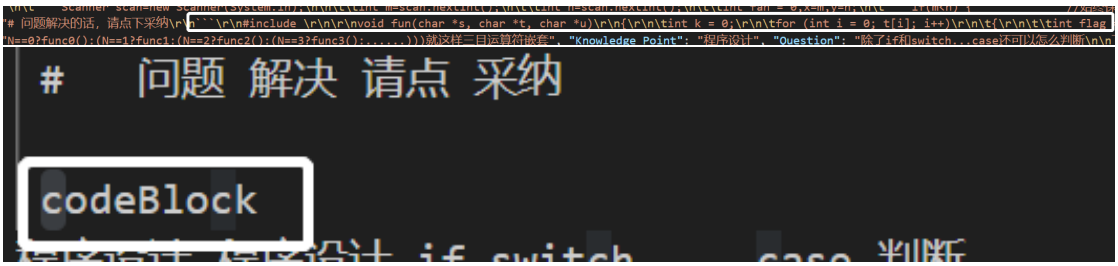


图 4: 用“‘区分代码块分词结果

2.2 word2vec

利用`gensim.models.Word2Vec`将自然语言转化为向量表示。

将gensim.models.Word2Vec在所有的数据的分词上训练，获得将词转化为对应向量表示的模型。

```
import gensim

# 用分词结果训练word2vec模型
# 根据需求配置 Word2Vec 参数并进行训练
model=gensim.models.Word2Vec(sentence,              # 分词结果
                             vector_size=128,   # 词向量维度
                             window=25,         # 相关前后文的窗口大小
                             min_count=10,      # 词频
                             workers=4,         # 线程数
                             hs=1,
                             negative=0,
                             epochs=20          # 迭代次数
                             )

model.save('1.model')
```

再将训练好的word2vec模型将训练集和测试集中的每个问答对转化为向量表示。我使用的转化方式是将每个问答对中在word2vec中出现的分词的向量和取平均。

```
from gensim.models import Word2Vec

# 导入训练好的模型
model=Word2Vec.load('1.model')

def get_sentence_vector(sentence):
    # 将句子用jieba分词
    word_list=[word for word in jieba.lcut(sentence) if word not in stopwords]
    # 所有能找到分词结果的词
    vectors = [model.wv[word] for word in word_list if word in model.wv]
    if vectors:
        return sum(vectors) / len(vectors)
    else:
        return average_vector
```

最后使用sklearn库的随机森林模型做分类训练。

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf_classifier = RandomForestClassifier(n_estimators=190,
                                       max_depth=27)
```

```
rf_classifier.fit(X_train, Y_train)

Y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
```

在试验过程中发现，word2vec的window参数对结果影响较大，其原因极有可能是适当增加窗口数，有利于word2vec模型学到词语的前后文信息。 word2vec的epoch参数对结果影响较大，增加迭代次数能提高模型准确率，但是迭代次数过多后，训练效率不高，还可能出现过拟合现象。增大随机森林的n_estimators参数对预测正确率影响较明显，但是增加到130以上后，结果不再有明显提升。

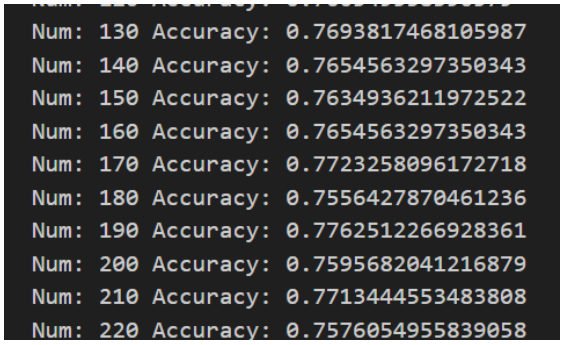


图 5: 改变n_estimators大小对正确率的影响

用jieba分词、gensim.models.Word2Vec转化词向量、sklearn.ensemble.RandomForestClassifier现分类任务，获得的最优正确率在0.77-0.78。

2.3 TfidfVectorizer

2.3.1 使用TfidfVectorizer词向量化

TfidfVectorizer是scikit-learn中的一个文本特征提取器，用于将文本转换为基于TF-IDF（Term Frequency-Inverse Document Frequency）的向量表示。

TF-IDF是一种常用的用于衡量文本中词语重要性的方法。它结合了两个指标：词频（Term Frequency，TF）和逆文档频率（Inverse Document Frequency，IDF）。TF衡量了在单个文档中词语出现的频率，而IDF衡量了该词语在整个文集中的重要程度。

为了保证在使用TfidfVectorizer时对中文词汇仍然能有好的分词效果，使用了自定义的分词工具，借用jieba分词的结果。

用TfidfVectorizer词向量化后的训练集训练随机森林，在测试集上测试正确率。最优结果在0.85-0.86

```
Num: 20 Accuracy: 0.8233562315996075
Num: 30 Accuracy: 0.8204121687929342
Num: 40 Accuracy: 0.844946025515211
Num: 50 Accuracy: 0.8351324828263003
Num: 60 Accuracy: 0.8429833169774289
Num: 70 Accuracy: 0.8537782139352306
Num: 80 Accuracy: 0.8429833169774289
Num: 90 Accuracy: 0.8370951913640824
Num: 100 Accuracy: 0.8498527968596663
Num: 110 Accuracy: 0.845927379784102
```

图 6: 改变n_estimators大小（max_depth=None）对正确率的影响

```
Num: 72 Accuracy: 0.8331697742885181
Num: 74 Accuracy: 0.8410206084396468
Num: 76 Accuracy: 0.845927379784102
Num: 78 Accuracy: 0.831207065750736
Num: 80 Accuracy: 0.8420019627085378
Num: 82 Accuracy: 0.844946025515211
Num: 84 Accuracy: 0.8557409224730128
Num: 86 Accuracy: 0.8527968596663396
Num: 88 Accuracy: 0.8400392541707556
Num: 90 Accuracy: 0.8400392541707556
Num: 92 Accuracy: 0.8341511285574092
```

图 7: 改变max_depth大小（n_estimators=70）对正确率的影响

发现使用TfidfVectorizer词向量化的效果明显优于word2vec，我认为是因为word2vec更关注词的上下文信息，而TfidfVectorizer用IDF衡量了该词语在整个文集中的重要程度，关注到了某些特殊的词在分类中的意义，其词向量化的结果对分类任务更加有利。

2.3.2 网格搜索

定义了网格搜索的参数空间，将TfidfVectorizer和随机森林模型用pipeline相连。创建GridSearchCV对象在训练集上执行网格搜索，获得结果最好的参数组合，使用最佳参数组合的模型在测试集上进行预测。

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.model_selection import GridSearchCV
```

```
from sklearn.pipeline import Pipeline

# 定义要搜索的参数空间
param_grid = {
    'vectorizer__max_features': [None,500,1000,1500,2000],
    'rf_model__n_estimators': [60,90,110],
    'rf_model__max_depth': [None,40,80]
}

# 构建机器学习工作流
pipeline=Pipeline([
    ('vectorizer',vectorizer),
    ('rf_model', rf_model)
])


# 创建GridSearchCV对象，指定pipeline和参数空间
grid_search = GridSearchCV(pipeline, param_grid=param_grid, scoring='accuracy', cv=3,
    refit=True)

# 在训练集上执行网格搜索
# 在训练集上划分“训练集”和“测试集”，反复验证，获得结果较好的参数
grid_search.fit(X_train, Y_train)
```

用上述参数空间，训练得到的在测试集上的正确率为0.8253189401373896，低于手动调参的正确率。我认为原因可能是使用网格搜索得到的最优参数组合是完全由训练集得出的，但是手动调参的参数是根据在测试集上的测试结果调整的，有“面向数据”之嫌。

2.4 sentence-bert

sentence-bert模型roberta-base-chinese-extractive-qa，损失函数BatchAllTripletLoss在测试集上迭代。使用预训练后的sentence-bert模型词向量化，再使用随机森林训练分类器。

原本想使用pipeline将使用的sentence-bert和随机森林连接，再使用损失函数训练，由于sentence-bert是基于深度学习的模型，并且其训练过程与传统的随机森林不兼容，因此无法直接在pipeline中拼接这两个步骤。

迭代十次后，预测正确率有一定的增长。我认为原因可能是sentence-bert模型在迭代后，能根据标签，学习到将被分为1和0的词区分的特征，因而词向量化的结果更有利

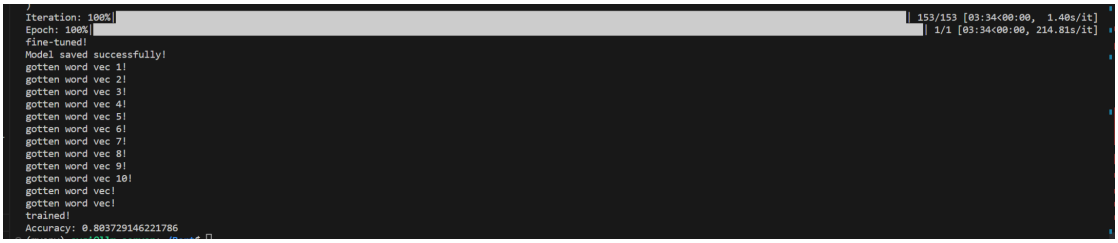


图 8: 迭代一次

图 9: 迭代十次

于分类。

2.5 bert分类

2.5.1 不改变bert参数

使用了bert-base-chinese的Tokenizer和预训练模型，将bert-base-chinese预训练模型和下游二分类任务连接。

```
#定义下游任务模型
#自定义操作，继承nn.Module类
class Model(torch.nn.Module):
    def __init__(self,pretrained) :
        super().__init__()
        # 入通道数，出通道数(0,1)
        self.linear_layer=torch.nn.Linear(768,2)
        self.pretrained=pretrained

    #实现模型的功能，实现各个层之间的连接关系的核心
    def forward(self,input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
        # 所有计算得出的tensor的requires_grad都自动设置为False
            out = self.pretrained(input_ids=input_ids,
                                  attention_mask=attention_mask,
```

```
                                token_type_ids=token_type_ids)


        # self.linear_layer执行forward函数
        out = self.linear_layer(out.last_hidden_state[:, 0])#用特征中的第0个来分类
        # 归一化指数函数（多个输出结点）
        out = out.softmax(dim=1)

        return out
```

选用优化算法和损失函数在训练集上训练模型。

```
from transformers import AdamW #优化算法，利于收敛

optimizer = AdamW(my_model.parameters(), lr=2e-4)#学习模型的参数，lr：学习率
criterion = torch.nn.CrossEntropyLoss()#损失函数
```

将训练得的模型在测试集上测试正确率。

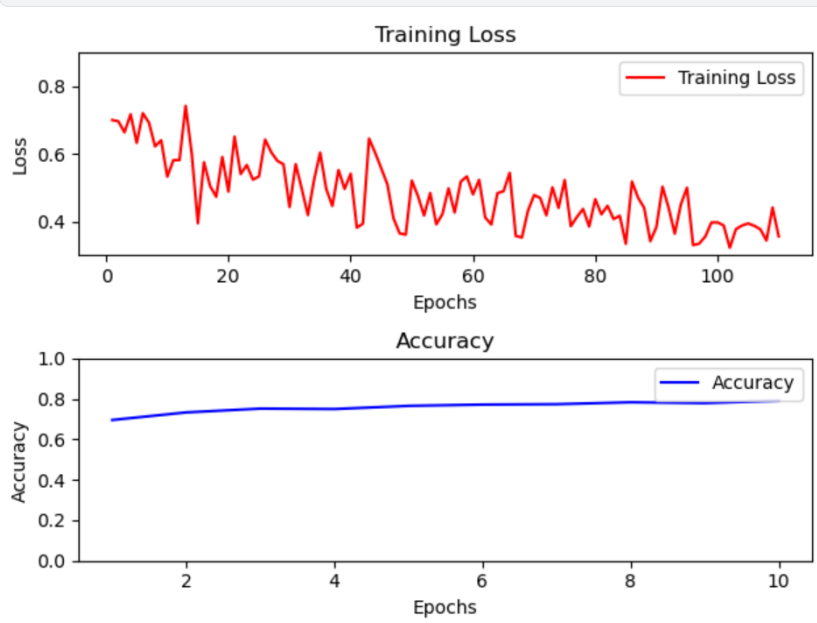
在调整参数过程中，发现由于训练集数据量小，适当减小batch_size能提升训练效果，但是过小的batch_size会导致loss出现大幅度的振荡。

由于bert预训练模型最多接受长度为512的文本，而我们的数据集为较长的问答对，直接截取问答对的前510个词无法较好提取出文本信息。为了较好表现问答对信息，我在构建数据集时，提取了Question头部32个词，尾部94个词，Answer的头部128个词，尾部256个词，拼接成字符串。这一个处理对训练效果的提升较明显。

```
    def __getitem__(self,ind):
        text=''
        if(len(self.dataset[ind]['Question'])<=126):
            text+=self.dataset[ind]['Question']
        else:
            text+=self.dataset[ind]['Question'][:32]
            text+=self.dataset[ind]['Question'][-94:]
        if(len(self.dataset[ind]['Answer'])<=384):
            text+=self.dataset[ind]['Answer']
        else:
            text+=self.dataset[ind]['Answer'][:128]
            text+=self.dataset[ind]['Answer'][-256:]
        label=self.dataset[ind]['Label']
        return text,label
```

学习率对训练效果的影响很大，在采用学习率为1-e3到1e-5，均出现了loss曲线高频振荡的情况，多次实验后，发现在取学习率为5e-6时，训练效果较好。

初始的模型，冻结了bert预训练模型的参数，仅更新下游模型线性层参数。通过解冻bert预训练模型的最后一层的参数，在训练时将bert预训练模型也加入更新。



[19]:
```
# 测试
test(my_model,Dataset('Test'),False)
```

```
100% 1/1 [00:00<00:00, 57.76it/s]
Accuracy: 0.8004032258064516
```

图 10: batch_size=16,lr=5e-6迭代十次训练结果

多次迭代，可以让模型学到更多东西，但是由于训练集小，迭代次数过多，会出现过拟合，在训练集上的正确率在0.9-1.0，但在测试集上的正确率在0.81波动。在下游模型的线性加dropout对过拟合有一定的规避效果。

2.6 投票法

将结果较好的模型作为基座，组建投票分类器。

我先选用了在使用相同模型结构训练，不同迭代次数的模型作为参数。每个模型的正确率如下表，最终投票结果在测试集上的结果为0.82。比原先的正确率都略有提升。由于模型结构相同，并不满足投票分类器用优而不同的弱分类器构建强分类器的要求，

效果不佳。

model	Accuracy
checkpoint10	0.813
checkpoint14	0.817
checkpoint18	0.814
checkpoint23	0.814
checkpoint26	0.815

表 1: 选用模型的正确率

A 模型参数

A.1 Word2Vec

gensim.models.Word2Vec参数

```
# 配置 Word2Vec 参数并进行训练
model=gensim.models.Word2Vec(sentence,
                             vector\_size=128,   # 词向量维度
                             window=25,         # 相关前后文的窗口大小
                             min\_count=10,      # 词频
                             workers=4,
                             hs=1,
                             negative=0,
                             epochs=20
                             )
```

使用Word2Vec模型词向量化，随机森林分类，随机森林参数

```
rf_classifier = RandomForestClassifier(n\_estimators=190,
                                       max\_depth=27)
```

A.2 TfidfVectorizer

```
def custom\_tokenizer(text):
    return text.split("。")

def get\_sentence\_token(sentence):
    # 将句子用jieba分词
```

```
    word\_list=[word for word in jieba.lcut(sentence) if word not in stopwords]
    # 将词以特殊的划分依据链接为string，用于作为TfidfVectorizer的样本
    str=""
    for word in word_list:
        str=str+word+"。"
    return str

vectorizer = TfidfVectorizer(tokenizer=custom_tokenizer)

rf_classifier = RandomForestClassifier(n_estimators=70)
```

B github仓库

爬虫、分类器训练等代码详见 https://github.com/Fangtangtang/CodeMate-PPCA-Project