|  |  |
| --- | --- |
| **自然语言处理及应用**  **实验报告** | |
|  | |
|  |  |
| **名称** | Text\_Classification(Bayes&RNN) |
| **姓名** | 卢佳源 |
| **班级** | 人工智能91 |
| **学号** | 2191121196 |
| Email | bjlujiayuan@126.com |
| **日期** | 2021-10-2 |

# 实验目的

按照新闻组进行分类标注。使用至少两种分类模型进行评估，5次交叉验证，并对分类结果进行分析讨论。数据集包含1000个文本，类属20个新闻组群里，总数20000个文档，数据文件夹名称为20\_newsgroups。

# 实验环境

OS：Linux操作系统的ubuntu20.04版本

IDE：VScode,anaconda3(‘base’:conda)

Terminal：gnome-terminal -t $TITLE -x

Language：python(3.8.8 64-bit)

PATH：朴素贝叶斯：/home/lujiayuan/python/ Neural\_Network/text\_classfication2.py

RNN：/home/lujiayuan/python/ Neural\_Network/CNN/dataloader1.py、model.py以及main.py

# 实验方法

1. 朴素贝叶斯：
2. 实验思路：
3. 预处理数据集：使用DataSet函数建立存储预处理结果的文件夹（按照原数据集的目录结构建立新文件夹），之后使用stpList函数导入停用词表，再调用preprocess1函数进行词干提取、大写字母转化为小写字母、删去停用词，并将每篇文章进行分词处理，最后得到每篇文章的关键词，放在和原数据集相同目录结构的之前新建的文件夹中；
4. 划分训练集和测试集：使用train\_test函数，将训练集和测试集划分为9：1的比例进行训练和测试（其中可以自己手动调整测试集的起始和终止位置，便于之后做至少5次的交叉验证，本次实验我做了10次交叉验证）；
5. 统计每一个已预处理文件中的单词数量：调用wordnum函数，计算每个单词在文章中出现次数的同时，计算它们出现的概率，并以dict类型返回这两个结果，即得到单词和出现次数或概率的词向量，也即是每个种类的文章向量。
6. 统计测试单词的平滑后的在训练集中出现概率：调用wordprob函数按照平滑（add-one）公式进行训练集每个单词概率的处理；
7. 调用朴素贝叶斯分类器进行文本分类：调用BYS函数，首先读取之前预处理好的测试集文本，查看每个测试文本单词在训练集中出现的概率，并选出最大的概率作为该测试文本的所属类别，并将分类结果写入result列表中；
8. 计算分类准确率：调用accuracy函数，比较result列表和实际测试集的类别（right列表）的重合程度，返回每次交叉验证的分类正确率；
9. 最后调用主函数，将上述过程逐一调用执行。
10. 实验代码：

import argparse

from genericpath import samefile

import tarfile

from nltk import probability

from nltk.sem.logic import read\_type

from nltk.util import pr

from numpy.random.mtrand import sample

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import os

import pandas as pd

import re

import nltk

from nltk.corpus import stopwords as pw

import numpy as np

from numpy import \*

import math

import operator

from operator import itemgetter, truth

from numpy import linalg

def DataSet(path,targetname):

# path="20\_newsgroups/"

srcData=os.listdir(path)

for i in range(len(srcData)):

srcDatadir=path+srcData[i]

srcDataList=os.listdir(srcDatadir)

targetdir=targetname+"/preprocess"+srcData[i]

if not os.path.exists(targetdir):

os.makedirs(targetdir)

else:

print('Exist')

for j in range(len(srcDataList)):

targetfile=targetname+'/preprocess'+srcData[i]+'/'+srcDataList[j]

srcfile=path+srcData[i]+'/'+srcDataList[j]

tarfiles=open(targetfile,'w',encoding='utf-8',errors='ignore')

datafiles=open(srcfile,'r',encoding='utf-8',errors='ignore').readlines()

for line in datafiles:

dataline=preprocess1(line)

for word in dataline:

tarfiles.write('%s\n'%word)

tarfiles.close()

print('%s%s'%(srcData[i],srcDataList[j]))

def DataSet1(path,targetname):

# path="20\_newsgroups/"

srcData=os.listdir(path)

for k in range(len(srcData)):

f1=path+srcData[k]

f2=os.listdir(f1)

tar=targetname+'/preprocess'+srcData[k]

for i in range(len(f2)):

srcDatadir=f1+'/'+f2[i]

srcDataList=os.listdir(srcDatadir)

targetdir=tar+'/preprocess'+f2[i]

if not os.path.exists(targetdir):

os.makedirs(targetdir)

else:

print('Exist')

for j in range(len(srcDataList)):

targetfile=targetdir+'/preprocess'+srcDataList[j]

srcfile=srcDatadir+'/'+srcDataList[j]

tarfiles=open(targetfile,'w',encoding='utf-8',errors='ignore')

datafiles=open(srcfile,'r',encoding='utf-8',errors='ignore').readlines()

for line in datafiles:

dataline=preprocess1(line)

for word in dataline:

tarfiles.write('%s\n'%word)

tarfiles.close()

print('%s%s'%(f2[i],srcDataList[j]))

def stpList(filepath):

stopwords=[line.split() for line in open(filepath,'r',encoding='utf-8',errors='ignore').readlines()]

return stopwords

def preprocess1(line):

stopword=stpList('stopword.txt')

cigan=nltk.PorterStemmer()

splits=re.compile('[^a-zA-Z]')

word=[cigan.stem(word.lower()) for word in splits.split(line) if len(word)>0 and word.lower() not in stopword]

word= [x for x in word if x!=' ']

return word

def IDF(filedir,filename):

# filedir='preprocess'

word\_doc={}

word\_IDF={}

tf={}

doc\_num=0.0

word\_List=os.listdir(filedir)

for k in range(len(word\_List)):

f1=filedir+word\_List[k]

f2=os.listdir(f1)

for i in range(len(f2)):

Sampledir=f1+'/'+f2[i]

Samplefile=os.listdir(Sampledir)

for j in range(len(Samplefile)):

word\_in\_doc\_num={}

word\_in\_doc\_sum=0

sample=Sampledir+'/'+Samplefile[j]

files=open(sample,'r',encoding='utf-8',errors='ignore').readlines()

for line in files:

word\_in\_doc\_sum+=1

word=line.strip('\n')

word\_in\_doc\_num[word]=word\_in\_doc\_num.get(word,0)+1

if word in word\_doc.keys():

word\_doc[word].add(Samplefile[j])

else:

word\_doc.setdefault(word,set())

word\_doc[word].add(Samplefile[j])

for word,num in word\_in\_doc\_num.items():

tf[word]=float(num)/float(word\_in\_doc\_sum)

print('finished ',i)

for i in word\_doc.keys():

doc\_num=len(word\_doc[i])

IDF=math.log(20017/doc\_num)/math.log(10)

word\_IDF[i]=IDF

IDFbook=open(filename,'w',encoding='utf-8',errors='ignore')

for word,IDFs in word\_IDF.items():

IDFbook.write('%s %.6f\n'%(word,IDFs))

IDFbook.close()

return word\_IDF

def train\_test(index0,right,trainpercent=0.9):

fr=open(right,'w',encoding='utf-8',errors='ignore')

fileDir = '20\_newsgroups'

featureList=os.listdir(fileDir)

for i in range(len(featureList)):

sampledir=fileDir+'/'+featureList[i]

samplefile=os.listdir(sampledir)

test\_index0=index0\*(len(samplefile)\*(1-trainpercent))

test\_indexn=(index0+1)\*(len(samplefile)\*(1-trainpercent))

for j in range(len(samplefile)):

if(j>=test\_index0)and(j<=test\_indexn):

fr.write('%s %s\n'%(samplefile[j],featureList[i]))

targetdir='Test\_Sample/TestSample'+str(index0)+'/'+featureList[i]

else:

targetdir='Train\_Sample/TrainSample'+str(index0)+'/'+featureList[i]

if not os.path.exists(targetdir):

os.makedirs(targetdir)

samdir=sampledir+'/'+samplefile[j]

sam=open(samdir,'r',encoding='utf-8',errors='ignore').readlines()

samw=open(targetdir+'/'+samplefile[j],'w',encoding='utf-8',errors='ignore')

for line in sam:

samw.write('%s\n'%line.strip('\n'))

samw.close()

DataSet1('Test\_Sample/','preprocess\_test')

DataSet1('Train\_Sample/','preprocess\_train')

IDF('preprocess\_test/','test\_idf')

IDF('preprocess\_train/','train\_idf')

def word2vec(trainsamdir,feature\_train):

IDF\_data={}

wordsum\_test=0.0

wordsum\_train=0.0

prob=0.0

condition={}

xianyan={}

tf\_idf={}

# trainsamdir='preprocess\_train/'

# testsamdir='preprocess\_test/'

# testidfdir='Test\_IDF'

trainidfdir='Train\_IDF'

# DataSet1(testsamdir,'preprocess\_test')

# DataSet1(trainsamdir,'preprocess\_train')

# IDF(testsamdir,'test\_idf')

fr\_test=open('test\_idf','r',encoding='utf-8',errors='ignore').readlines()

IDF\_test={}

for line in fr\_test:

v=line.strip('\n').split(' ')

word=v[0]

IDFs=v[1]

wordsum\_test+=1

IDF\_test[word]=IDFs

# IDF(trainsamdir,'train\_idf')

fr\_train=open('train\_idf','r',encoding='utf-8',errors='ignore').readlines()

IDF\_train={}

for line in fr\_train:

v1=line.strip('\n').split(' ')

word=v1[0]

IDFs1=v1[1]

wordsum\_train+=1

IDF\_train[word]=IDFs1

Trainw = open(trainidfdir, 'w',encoding='utf-8',errors='ignore')

feature\_train=os.listdir(trainsamdir)

for i in range(len(feature\_train)):

sampledir2=trainsamdir+'/'+feature\_train[i]

samplefile2=os.listdir(sampledir2)

for j in range(len(samplefile2)):

word\_in\_doc\_num1={}

word\_in\_doc\_sum1=0

samples1=sampledir2+'/'+samplefile2[j]

fr1=open(samples1,'r',encoding='utf-8',errors='ignore').readlines()

for line in fr1:

word\_in\_doc\_sum1+=1

word1=line.strip('\n')#.split(' ')

word\_in\_doc\_num1[word1]=word\_in\_doc\_num1.get(word1,0)+1

for word,num in word\_in\_doc\_num1.items():

# word=word.split(' ')

TF=float(num)/float(word\_in\_doc\_sum1)

# tf\_idf[word]=TF\*float(IDF\_train.get(word,0))

condition[word]=(float(num)+0.0001)/(float(word\_in\_doc\_sum1)+wordsum\_train)

xianyan[word]=(float(word\_in\_doc\_sum1))/float(wordsum\_train)

prob=float(condition[word]/(20\*xianyan[word]))

Trainw.write('%d %f %d %s %s %s'%(float(num),prob,float(word\_in\_doc\_sum1),feature\_train[i],samplefile2[j],word))

Trainw.write('\n')

return prob

print('finishtrain',j)

Trainw.close()

def wordnum(dir):

word\_doc\_num={}

word\_doc\_prob={}

attdir=os.listdir(dir)

for i in range(len(attdir)):

cnt=0

filedir=dir+'/'+attdir[i]

file=os.listdir(filedir)

for j in range(len(file)):

samfile=filedir+'/'+file[j]

word=open(samfile,'r',encoding='utf-8',errors='ignore').readlines()

for k in word:

cnt+=1

words=k.strip('\n')

name=attdir[i]+'\_'+words

word\_doc\_prob[name]=word\_doc\_prob.get(name,0)+1

word\_doc\_num[attdir[i]]=cnt

return word\_doc\_prob,word\_doc\_num

def wordprob(trainpath,testword,wordnum,wordsum,word\_doc\_prob):

prob=0.0

word\_doc\_num=wordnum[trainpath]

for i in range(len(testword)):

name=trainpath+'\_'+testword[i]

if name in word\_doc\_prob:

condition\_fenzi=word\_doc\_prob[name]+1

else:

condition\_fenzi=0.0+1

for j in range(20):

word\_doc\_num1=float(wordnum.get(j,0))+float(''.join('%s' %id for id in wordsum))

xianyan=math.log(condition\_fenzi/word\_doc\_num1)

prob=prob+xianyan

probabilitys=prob+math.log(word\_doc\_num)-math.log(float(''.join('%s' %id for id in wordsum)))

return probabilitys

def BYS(trainpath,testpath,result):

resultw=open(result,'w',encoding='utf-8',errors='ignore')

word\_doc\_prob,word\_doc\_num=wordnum(trainpath)

wordsum=sum(word\_doc\_num.values())

testfile=os.listdir(testpath)

for i in range(len(testfile)):

testdir=testpath+'/'+testfile[i]

testsam=os.listdir(testdir)

for j in range(len(testsam)):

testword=[]

testsamdir=testdir+'/'+testsam[j]

lines=open(testsamdir,'r',encoding='utf-8',errors='ignore').readlines()

for line in lines:

word=line.strip('\n')

testword.append(word)

Pmax=0.0

traindir=os.listdir(trainpath)

for k in range(len(traindir)):

P=wordprob(traindir[k],testword,word\_doc\_num,wordsum,word\_doc\_prob)

if k==0:

Pmax=P

attribute=traindir[k]

continue

if P>Pmax:

Pmax=P

attribute=traindir[k]

resultw.write('%s %s\n'%(testsam[j],attribute))

resultw.close()

def accuracy(right,result,k):

rightdict={}

resultdict={}

rightcnt=0.0

for i in open(right,'r',encoding='utf-8',errors='ignore').readlines():

(file,attribute)=i.strip('\n').split(' ')

rightdict[file]=attribute

for i in open(result,'r',encoding='utf-8',errors='ignore').readlines():

(file,attribute)=i.strip('\n').split(' ')

resultdict[file]=attribute

for file in rightdict.keys():

if(rightdict.get(file,0)==resultdict.get(file,0)):

rightcnt+=1.0

print('rightcnt: %d right: %d'%(rightcnt,len(rightdict)))

acc=rightcnt/len(rightdict)

print('accuracy %d : %f'%(k,acc))

return acc

def main():

# DataSet('20\_newsgroups/','preprocess')

# IDF()

acc=[]

for i in range(10):

right='right'+str(i)+'.txt'

train\_test(i,right)

for i in range(10):

trainpath='Train\_Sample/TrainSample'+str(i)

testpath='Test\_Sample/TestSample'+str(i)

result='result'+str(i)+'.txt'

BYS(trainpath,testpath,result)

for i in range(10):

right1='right'+str(i)+'.txt'

result='result'+str(i)+'.txt'

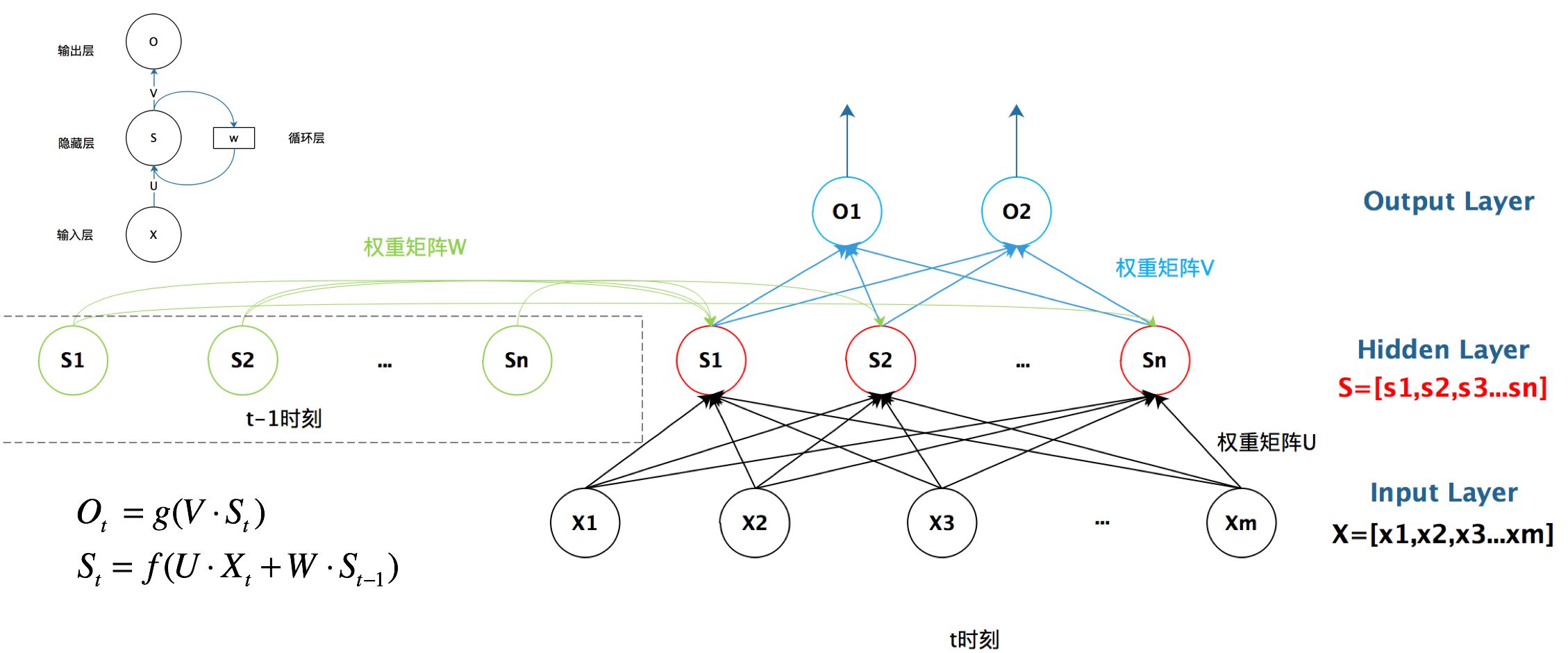
acc.append(accuracy(right1,result,i))

return acc

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. RNN：
2. 实验思路：
3. dataloader1.py:预处理部分代码，步骤与贝叶斯相似，但需要把单词转化为数值形式：
4. 准备数据：利用prepare\_data函数导入贝叶斯方法中预处理好的数据文件夹并屏蔽文件头；
5. Language类:建立词典并根据glove.6B.50d.txt将其转化为数值型列表；
6. 定义一个文本分类的数据集类；
7. 定义一个batch里面数据的组织方式，调用pad\_sequence包；
8. 读到glove.6B.50d.txt中已经与训练的词向量，并显示总共找到的词向量个数；
9. 建立dataloader（此函数要被main.py调用）：即将上述过程依次在此函数中执行，并且利用train\_test\_split包进行训练集和测试集的划分，最终返回的变量可以直接作为RNN的输入；
10. model.py:RNN模型建立：
11. 定义RNN类的模型：



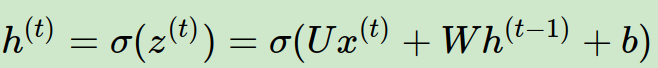
1. 初始化：

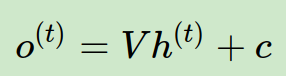
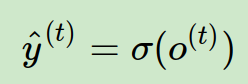
定义了输入规模，隐藏层大小，文本类别数、embedding层（定义编码维度——即多少个数字表示一个单词），并搭建pytorch的nn.RNN（）和设置网络中的全连接层来定义模型。



（[batch\_size, input\_dim] \* [input\_dim, num\_hiddens] + [batch\_size, num\_hiddens] \*[num\_hiddens, num\_hiddens] +bias）

1. 前向传播：

-->

-->

1. main.py:进行RNN模型训练和结果输出。
2. 初始化学习速率、训练次数、文本类别数；
3. 从dataloader中传入已经预处理好的文本数据；
4. 对数据进行训练：在每一次训练中，先启用batch normalization和drop out，将输入x放入RNN的模型中，建立优化器，并用交叉熵来计算损失函数，用梯度下降法减小损失函数；
5. 计算并打印分类准确度。
6. 实验代码：
7. dataloader1.py:

import os

import torch

import sys

from sklearn.model\_selection import train\_test\_split

from torch.nn.utils.rnn import pad\_sequence

from torch.utils.data import Dataset, DataLoader

data\_dir = "/home/lujiayuan/python/Neural\_Network/CNN"

text\_dir = os.path.join(data\_dir, '../preprocess')

file = []

labels\_index = {}

label = []

def prepare\_data(path, file, label):

file = []

label\_index = {}

label = []

for name in sorted(os.listdir(text\_dir)):

path = os.path.join(text\_dir, name)

# print(path)

if os.path.isdir(path):

label\_id = len(labels\_index)

label\_index[name] = label\_id

for fname in sorted(os.listdir(path)):

if fname.isdigit():

fpath = os.path.join(path, fname)

args = {} if sys.version\_info < (3,) else {'encoding': 'utf-8'}

with open(fpath, \*\*args) as f:

t = f.read()

i = t.find('\n\n') #屏蔽文件头

if 0 < i:

t = t[i:]

file.append(t)

label.append(label\_id)

return file,label

class Language:

def \_\_init\_\_(self):

self.word2id = {}

self.id2word = {}

def fit(self, sent\_list):

vocab = set()

for sent in sent\_list:

vocab.update(sent.split(" "))

word\_list = ["<pad>", "<unk>"] + list(vocab)

self.word2id = {word: i for i, word in enumerate(word\_list)}

self.id2word = {i: word for i, word in enumerate(word\_list)}

def transform(self, sent\_list, reverse=False):

sent\_list\_id = []

word\_mapper = self.word2id if not reverse else self.id2word

unk = self.word2id["<unk>"] if not reverse else None

for sent in sent\_list:

sent\_id = list(map(lambda x: word\_mapper.get(x, unk), sent.split(" ") if not reverse else sent))

sent\_list\_id.append(sent\_id)

return sent\_list\_id

class DatasetClass(Dataset):

def \_\_init\_\_(self, sents, labels):

self.sents = sents

self.labels = labels

def \_\_getitem\_\_(self, item):

return self.sents[item], self.labels[item]

def \_\_len\_\_(self):

return len(self.sents)

def batchorg(batch\_data):

batch\_data.sort(key=lambda data\_pair: len(data\_pair[0]), reverse=True)

sents, labels = zip(\*batch\_data)

sents\_len = [len(sent) for sent in sents]

sents = [torch.LongTensor(sent) for sent in sents]

padded\_sents = pad\_sequence(sents, batch\_first=True, padding\_value=0)

return torch.LongTensor(padded\_sents), torch.LongTensor(labels), torch.FloatTensor(sents\_len)

def word2vec(word2id, vec\_file\_path, vec\_dim=50):

word\_vectors = torch.nn.init.xavier\_uniform\_(torch.empty(len(word2id), vec\_dim))

word\_vectors[0, :] = 0

found = 0

with open(vec\_file\_path, "r", encoding="utf-8") as f:

lines = f.readlines()

for line in lines:

splited = line.split(" ")

if splited[0] in word2id:

found += 1

word\_vectors[word2id[splited[0]]] = torch.tensor(list(map(lambda x: float(x), splited[1:])))

if found == len(word2id) - 1:

break

return word\_vectors.float()

def dataloader(dataset\_path="../20\_newsgroups", sent\_name="Phrase", label\_name="Sentiment", batch\_size=32, vec\_file\_path="glove.6B.50d.txt", debug=False):

X, y = prepare\_data(path=dataset\_path, file=sent\_name, label=label\_name)

if debug:

X, y = X[:100], y[:100]

X\_language = Language()

X\_language.fit(X)

X = X\_language.transform(X)

word\_vectors = word2vec(X\_language.word2id, vec\_file\_path=vec\_file\_path, vec\_dim=50)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)

train\_dataset, test\_dataset = DatasetClass(X\_train, y\_train), DatasetClass(X\_test, y\_test)

train\_dataloader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, collate\_fn=batchorg)

test\_dataloader = DataLoader(test\_dataset, batch\_size=batch\_size, collate\_fn=batchorg)

return train\_dataloader, test\_dataloader, word\_vectors, X\_language

1. model.py:

from itertools import tee

from numpy.core.fromnumeric import shape

from torch import tensor

import torch.nn as nn

import torch

from dataloader1 import dataloader

class myRNN(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_size, num\_of\_class, weights=None):

super(myRNN, self).\_\_init\_\_()

self.vocab\_size = vocab\_size

self.hidden\_size = hidden\_size

self.num\_of\_class = num\_of\_class

self.embedding\_dim = embedding\_dim

if weights is not None:

self.embed = nn.Embedding(num\_embeddings=vocab\_size, embedding\_dim=embedding\_dim, \_weight=weights)#创建一个词嵌入模型

else:

self.embed = nn.Embedding(num\_embeddings=vocab\_size, embedding\_dim=embedding\_dim)

self.rnn = nn.RNN(input\_size=embedding\_dim, hidden\_size=hidden\_size, batch\_first=True)#搭建基于序列的循环神经网络

self.hidden2label = nn.Linear(hidden\_size, num\_of\_class)#用于设置网络中的全连接层

def forward(self, input\_sents):

batch\_size, \_ = input\_sents.shape

embedding\_out = self.embed(input\_sents)

h0 = torch.randn(1, batch\_size, self.hidden\_size)#用来生成随机数字的tensor，这些随机数字满足标准正态分布（0~1）

\_, hn = self.rnn(embedding\_out, h0)

hidlable = self.hidden2label(hn).squeeze(0)

return hidlable

1. main.py:

from torch import optim

import torch

import sys

sys.path.append('/home/lujiayuan/python/Neural\_Network/CNN')

from model import myRNN

from dataloader1 import dataloader

import numpy as np

if \_\_name\_\_ == "\_\_main\_\_":

learning\_rate = 0.001

epoch\_num = 1000

num\_of\_class = 20

train=0

test=0

train\_iter, test\_iter, word\_vectors, X\_lang = dataloader(batch\_size=1000, debug=True)

model = myRNN(vocab\_size=len(word\_vectors), embedding\_dim=300, hidden\_size=128, num\_of\_class=num\_of\_class, weights=word\_vectors)

optimizer = optim.Adam(model.parameters(), lr=learning\_rate)#用来保存当前的状态，并能够根据计算得到的梯度来更新参数。

loss\_fun = torch.nn.CrossEntropyLoss()

for epoch in range(epoch\_num):

model.train() #启用batch normalization和drop out # 包含dropout或者BN的模型需要指定

for i, batch in enumerate(train\_iter):

x, y, lens = batch

#梯度下降法

hidlable = model(x)

optimizer.zero\_grad()#清空过往梯度 #是把梯度置零，也就是把loss关于weight的导数变成0

loss = loss\_fun(hidlable, y)

loss.backward()#反向传播，计算当前梯度

optimizer.step()#根据梯度更新网络参数

# # gradient descent

# weights = [0] \* n

# alpha = 0.0001

# max\_Iter = 50000

# for i in range(max\_Iter):

# loss = 0

# d\_weights = [0] \* n

# for k in range(m):

# h = dot(input[k], weights)

# d\_weights = [d\_weights[j] + (label[k] - h) \* input[k][j] for j in range(n)]

# loss += (label[k] - h) \* (label[k] - h) / 2

# d\_weights = [d\_weights[k]/m for k in range(n)]

# weights = [weights[k] + alpha \* d\_weights[k] for k in range(n)]

# if i%10000 == 0:

# print ("Iteration %d loss: %f"%(i, loss/m))

# print (weights)

model.eval()

'''

不启用 Batch Normalization 和 Dropout。

如果模型中有BN层(Batch Normalization）和Dropout，在测试时添加model.eval()。model.eval()是保证BN层能够用全部训练数据的均值和方差，即测试过程中要保证BN层的均值和方差不变。对于Dropout，model.eval()是利用到了所有网络连接，即不进行随机舍弃神经元。

训练完train样本后，生成的模型model要用来测试样本。在model(test)之前，需要加上model.eval()，否则的话，有输入数据，即使不训练，它也会改变权值。这是model中含有BN层和Dropout所带来的的性质。

'''

train\_accs = []

for i, batch in enumerate(train\_iter):

x, y, lens = batch

\_, y\_pre = torch.max(hidlable, -1)

acc = torch.mean((torch.tensor(y\_pre == y, dtype=torch.float)))

train\_accs.append(acc)

train\_acc = np.array(train\_accs).mean()

test\_accs = []

for i, batch in enumerate(test\_iter):

x, y, lens = batch

hidlable = model(x)

\_, y\_pre = torch.max(hidlable, -1)

acc = torch.mean((torch.tensor(y\_pre == y, dtype=torch.float)))

test\_accs.append(acc)

test\_acc = np.array(test\_accs).mean()

print("epoch %d train acc:%.6f, test acc:%.6f" % (epoch, train\_acc, test\_acc))

train+=train\_acc

test+=test\_acc

if train\_acc >= 0.9999:

break

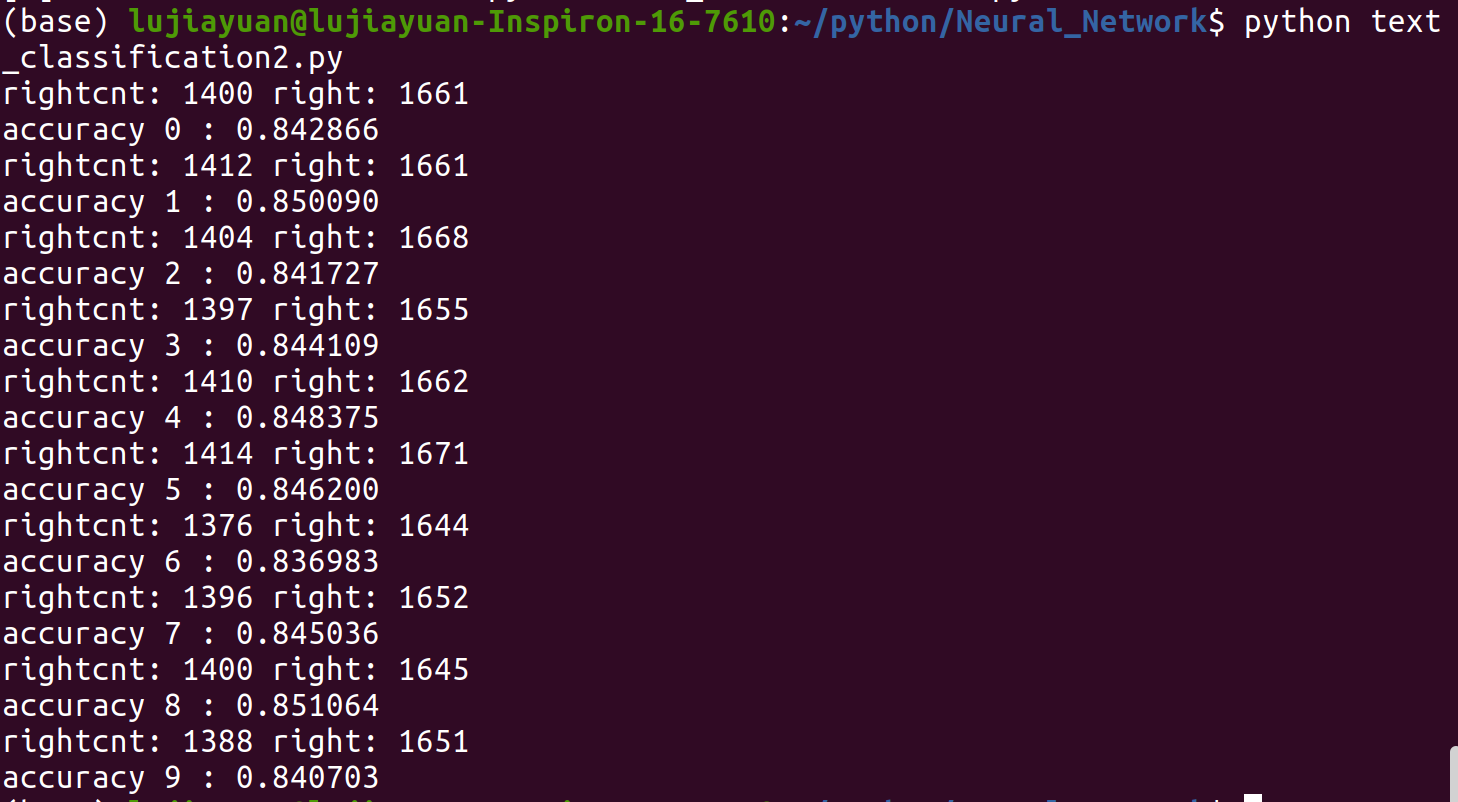
train=train/epoch

test=test/epoch

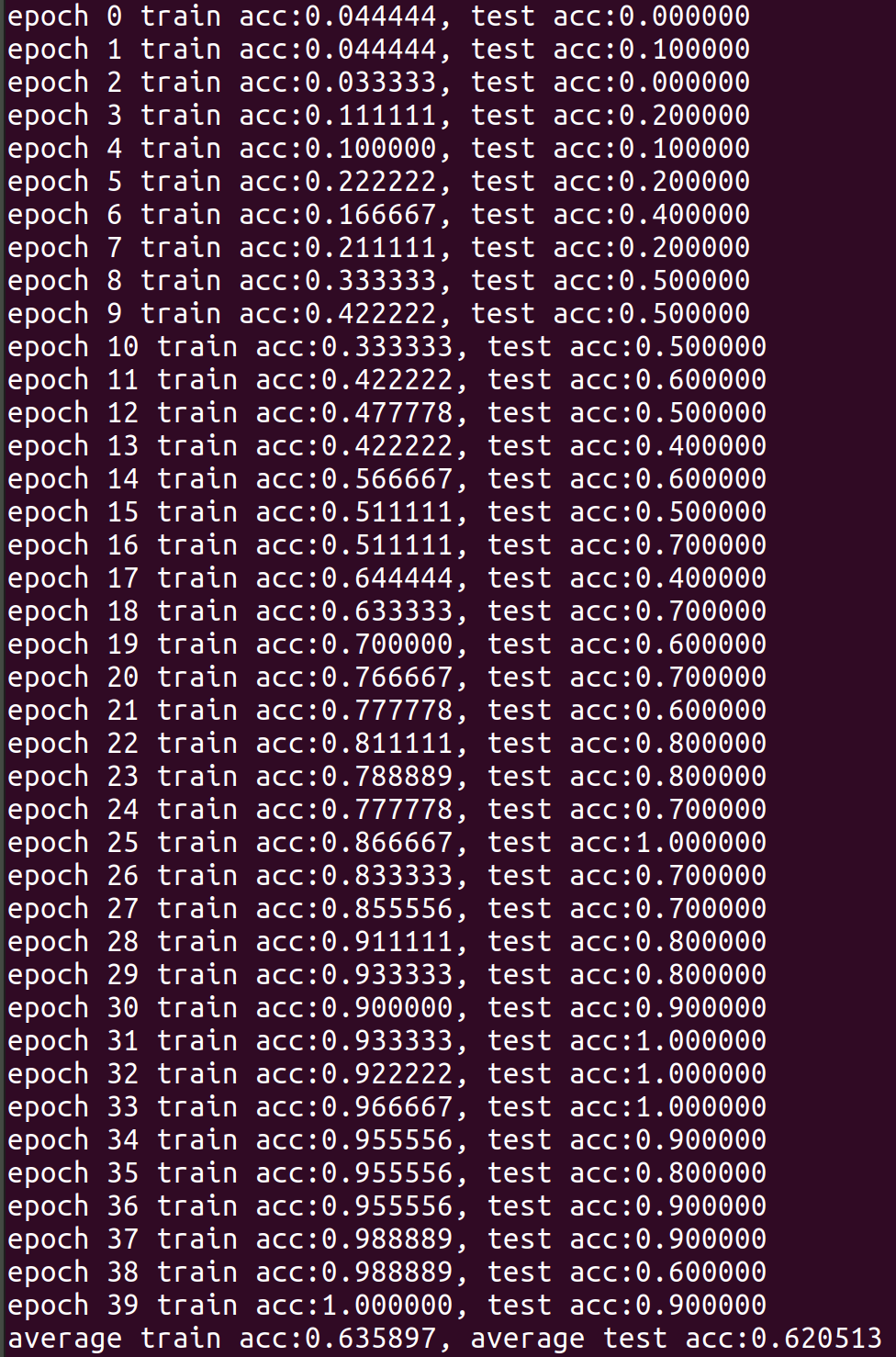
print("average train acc:%.6f, average test acc:%.6f" % (train, test))

1. **实验结果**
2. 实验
3. 结果截图：

朴素贝叶斯：



RNN：



1. 实验代码解析：
2. 朴素贝叶斯：打印结果中，rightcnt表示分类器分类结果和测试文本真正所属类别的重合个数，right表示测试文本真正所属类别而进行测试的测试文本总数，accuracy表示本次交叉验证中分类器的准确率。
3. RNN：打印的结果中显示了训练次数、训练集准确率和测试集准确率，以及所有训练之后的准确率的平均值。
4. 两种方法准确率比较：
5. 朴素贝叶斯：文本分类正确率大约在84.5%；
6. RNN：文本分类正确率大约在62%；
7. 原因分析：
8. 预处理过程不同，进入到分类器的数据类型是不一样的，贝叶斯进入分类器的是词和词频，RNN进入分类器的是将文本转化为数值类型，转化过程依赖已训练好的向量文本glove.6B.50d.txt。
9. 分类器分类效果不同：贝叶斯为线性分类器，基于变量是相互独立的，而RNN为非线性分类器，是一种循环反馈机制。
10. 交叉验证结果：
11. 贝叶斯：通过训练集的不同起始索引来划分训练集和测试集，共进行了10次交叉验证，即结果截图中的10个结果；
12. RNN：通过train\_test\_split（）函数将原始数据按照比列随即划分为训练集和测试集，每一次训练的测试集和训练集均不同，可以认为是相互验证的结果。
13. **遇到问题及解决思路**

问题1：在朴素贝叶斯分类器的构建过程中，我把预处理的调用写在了主函数中，但并没有在第一次预处理完注释掉，导致刚开始每一次运行代码都要等一遍预处理，十分浪费时间；

解决思路：在划分好训练集和测试集后，调用预处理，得到预处理后的文件夹后就将预处理步骤注释掉，直接观察分类结果；

问题2：刚开始我总是想办法把贝叶斯和RNN方法的预处理做成相同步骤，这样在两种方法比较时才有可比性，但是之后我发现，想要用到神经网络来做分类，最好将文本提取的单词接着化成数值向量形式作为输入，更方便来构造和训练神经网络。

解决思路：使用不同的预处理方法，针对不同的分类器，预处理的方法会有所不同，要找到最合适的预处理方式。

问题3：在写RNN过程中，没有弄清楚反向传播的作用以及反向传播的过程（公式）。

解决思路：反向传播即通过梯度下降法进行每一次的迭代，得到合适的模型参数，其公式推导可以从一些教程上学习到。