形状, 圆圈

描述已自动生成

学 期 2021-2022（2）

**卡通人物

中度可信度描述已自动生成**

**深度学习与自然语言处理第三次大作业**

基于LDA模型对于文本建模

|  |  |
| --- | --- |
| 院（系）名称 | 自动化科学与电气工程学院 |
| 专业名称 | 电子信息 |
| 学生姓名 | 熊方书 |
| 学号 | ZY2103309 |
| 指导老师 | 秦曾昌 |

2022年 5月

# 引言

## 问题描述

从给定的语料库中均匀抽取200个段落（每个段落大于500个词）， 每个段落的标签就是对应段落所属的小说。利用LDA模型对于文本建模，并把每个段落表示为主题分布后进行分类。验证与分析分类结果。

## LDA模型简介

**1.2.1 简介**

在机器学习领域，LDA是两个常用模型的简称：Linear Discriminant Analysis 和 Latent Dirichlet Allocation。本文的LDA仅指代Latent Dirichlet Allocation. LDA 在主题模型中占有非常重要的地位，常用来文本分类。

**1.2.2 词袋模型与文本建模**

LDA 采用词袋模型。所谓词袋模型，是将一篇文档，我们仅考虑一个词汇是否出现，而不考虑其出现的顺序。在词袋模型中，“我喜欢你”和“你喜欢我”是等价的。一篇文档，可以看成是一组有序的词的序列。从统计学角度来看，文档的生成可以看成是上帝抛掷骰子生成的结果，每一次抛掷骰子都生成一个词汇，抛掷N词生成一篇文档。在统计文本建模中，我们希望猜测出上帝是如何玩这个游戏的，这会涉及到两个最核心的问题：第一个问题就是模型中都有哪些参数；第二个问题就生成词序列的规则是什么。

**1.2.3 LDA生成文档的过程**

1. 按照先验概率选择一篇文档

2. 从Dirichlet分布中取样生成文档的主题分布，主题分布由超参数为的Dirichlet分布生成

3. 从主题的多项式分布中取样生成文档第j个词的主题

4. 从Dirichlet分布中取样生成主题对应的词语分布，词语分布由参数为的Dirichlet分布生成

5. 从词语的多项式分布中采样最终生成词语

**1.2.4 LDA模型的数学基础**

1、gamma函数



性质：1、

2、

2、二项分布

二项分布是由n个独立的是、非重复实验中成功次数的离散概率分布其中每次成功的概率为p，相当于你去求婚，每次求婚都有两种结果，成功或失败，如果实验一次，则称为伯努利分布，如果重复实验n次的话，则称为二项分布，记为：X ~ B(n,p),它的概率密度函数为：



3、多项式分布



4、Beta分布



5、狄利克雷分布



其中：



## 1.2实验流程



## 代码设计

使用Python语言完成算法和整个实验流程。代码输入输出如下所示：

详细代码见附录。

# 结果分析与总结

## 实验结果



## 实验分析

由上图可见，训练所得到的LDA模型的效果较好，准确性较高

# 3代码

import jieba

import os

import re

import time

import math

import numpy as np

import random

from gensim import corpora, models

from collections import defaultdict

from sklearn.svm import SVC

def data\_preprocessing(data\_roots, abandon\_stop\_words):

listdir = os.listdir(data\_roots)

char\_to\_be\_replaced = "\n `1234567890-=/\*-~!@#$%^&\*()\_+qwertyuiop[]\\QWERTYUIOP{}|asdfghjkl;" \

"'ASDFGHJKL:\"zxcvbnm,./ZXCVBNM<>?~！@#￥%……&\*（）——+【】：；“‘’”《》？，。" \

"、★「」『』～＂□ａｎｔｉ－ｃｌｉｍａｘ＋．／０１２３４５６７８９＜＝＞＠Ａ" \

"ＢＣＤＥＦＧＨＩＪＫＬＭＮＯＰＱＲＳＴＶＷＸＹＺ［＼］ｂｄｅｆｇｈｊｋｏｐｒｓ" \

"ｕｖｗｙｚ￣\u3000\x1a"

char\_to\_be\_replaced = list(char\_to\_be\_replaced)

txt\_corpus = []

label\_idxes = []

label\_words = []

label\_idx = 0

label\_idx\_to\_words = dict()

stop\_words\_list = []

for tmp\_file\_name in os.listdir("./data1/"): # replace this path with the stopwords path

with open("./data1/"+tmp\_file\_name, "r", encoding="utf-8", errors="ignore") as f:

stop\_words\_list.extend([word.strip('\n') for word in f.readlines()])

for tmp\_file\_name in listdir:

if tmp\_file\_name == "inf.txt":

continue

path = os.path.join(data\_roots, tmp\_file\_name)

if os.path.isfile(path):

with open(path, "r", encoding="gbk", errors="ignore") as tmp\_file:

tmp\_file\_context = tmp\_file.read()

for tmp\_char in char\_to\_be\_replaced:

tmp\_file\_context = tmp\_file\_context.replace(tmp\_char, "")

tmp\_file\_context = tmp\_file\_context.replace("本书来自免费小说下载站更多更新免费电子书请关注", "")

if abandon\_stop\_words:

for tmp\_char in stop\_words\_list:

tmp\_file\_context = tmp\_file\_context.replace(tmp\_char, "")

txt\_corpus.append(tmp\_file\_context)

label\_idxes.append(label\_idx)

label\_words.append(tmp\_file\_name.split(".txt")[0])

label\_idx\_to\_words[label\_idx] = tmp\_file\_name.split(".txt")[0]

label\_idx += 1

return txt\_corpus, label\_idxes, label\_words, label\_idx\_to\_words

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

num\_topics = 50

num\_docs = 200

len\_per\_doc = 500

abandon\_stop\_words = True

print("主题数：{}，段落(文档)数：{}，每段话字数：{}，是否去除停用词：{}".format(num\_topics, num\_docs, len\_per\_doc, "yes" if abandon\_stop\_words else "no"))

print("Preparing data...")

data\_roots = './data1/' # replace this path with the txt files path

txt\_corpus, label\_idxes, label\_words, label\_idx\_to\_words = data\_preprocessing(data\_roots, abandon\_stop\_words)

whole\_samples = []

#### get training samples and testing samples

for i in range(len(txt\_corpus)):

for j in range(num\_docs//len(txt\_corpus) + 1):

tmp\_start = random.randint(0, len(txt\_corpus[i])-len\_per\_doc-1)

tmp\_sample = list(jieba.cut(txt\_corpus[i][tmp\_start:tmp\_start + len\_per\_doc]))

whole\_samples.append((label\_idxes[i], tmp\_sample))

random.shuffle(whole\_samples)

whole\_samples = whole\_samples[:num\_docs]

train\_data, train\_label = [], []

test\_data, test\_label = [], []

for i in range(int(len(whole\_samples) \* (1 - 0.2))):

train\_data.append(whole\_samples[i][1])

train\_label.append(whole\_samples[i][0])

for i in range(int(len(whole\_samples) \* (1 - 0.2)), len(whole\_samples)):

test\_data.append(whole\_samples[i][1])

test\_label.append(whole\_samples[i][0])

#### train lda

dictionary = corpora.Dictionary(train\_data)

lda\_corpus\_train = [dictionary.doc2bow(tmp\_doc) for tmp\_doc in train\_data]

print("Trainng LDA model...")

lda = models.LdaModel(corpus=lda\_corpus\_train, id2word=dictionary, num\_topics=num\_topics)

#### train svm classifier for correct label

train\_topic\_distribution = lda.get\_document\_topics(lda\_corpus\_train)

train\_features = np.zeros((len(train\_data), num\_topics))

for i in range(len(train\_topic\_distribution)):

tmp\_topic\_distribution = train\_topic\_distribution[i]

for j in range(len(tmp\_topic\_distribution)):

train\_features[i][tmp\_topic\_distribution[j][0]] = tmp\_topic\_distribution[j][1]

print("Training SVM classifier...")

assert len(train\_label) == len(train\_features)

train\_label = np.array(train\_label)

classifier = SVC(kernel='linear', probability=True)

classifier.fit(train\_features, train\_label)

print("Prediction accuracy of training samples is {:.4f}.".format(sum(classifier.predict(train\_features) == train\_label) / len(train\_label)))

#### testing

lda\_corpus\_test = [dictionary.doc2bow(tmp\_doc) for tmp\_doc in test\_data]

test\_topic\_distribution = lda.get\_document\_topics(lda\_corpus\_test)

test\_features = np.zeros((len(test\_data), num\_topics))

for i in range(len(test\_topic\_distribution)):

tmp\_topic\_distribution = test\_topic\_distribution[i]

for j in range(len(tmp\_topic\_distribution)):

test\_features[i][tmp\_topic\_distribution[j][0]] = tmp\_topic\_distribution[j][1]

assert len(test\_label) == len(test\_features)

test\_label = np.array(test\_label)

print("Prediction accuracy of testing samples is {:.4f}.".format(sum(classifier.predict(test\_features) == test\_label) / len(test\_label)))