成 绩 _____

北京航空航天大學
BEIHANG UNIVERSITY

# 深度学习与自然语言处理

# 第 3 次大作业

## LDA 主题模型进行文本分类



| | |
|---|---|
| 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 号 | ZY2103113 |
| 姓 名 | 孙茗逸 |
| 指 导 教 师 | 秦曾昌 |



2022 年 5 月 6 日

## 一、任务描述

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。

## 二、实验原理

### 1. LDA 主题模型

在文本挖掘领域中大量的数据都是非结构化的，难以从信息中直接获取相关和期望的信息。主题模型（Topic Model）能够识别在文档里的主题，并且挖掘语料里隐藏信息，在主题聚合、特征选择等场景有广泛的用途。

LDA（Latent Dirichlet Allocation）是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。所谓生成模型，就是说，我们认为一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语""这样一个过程得到。文档到主题服从多项式分布，主题到词服从多项式分布。

LDA 采用了“词袋”的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是没有考虑词与词之间的顺序，这简化了问题的复杂性。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布。

1) 按照先验概率 $P(d_i)$ 选择一篇文档 $d_i$

2) 从狄利克雷分布 $\alpha$ 中取样生成文档 $d_i$ 的主题分布 $\theta_i$

3) 从主题的多项式分布 $\theta_i$ 中取样生成文档 $d_i$ 第 j 个词的主题 $z_{i,j}$

4) 从狄利克雷分布 β 中取样生成主题 $z_{i,j}$ 对应的词语分布 $\phi_{z_{i,j}}$

5) 从词语的多项式分布 $\phi_{z_{i,j}}$，中采样最终生成词语 $w_{i,j}$

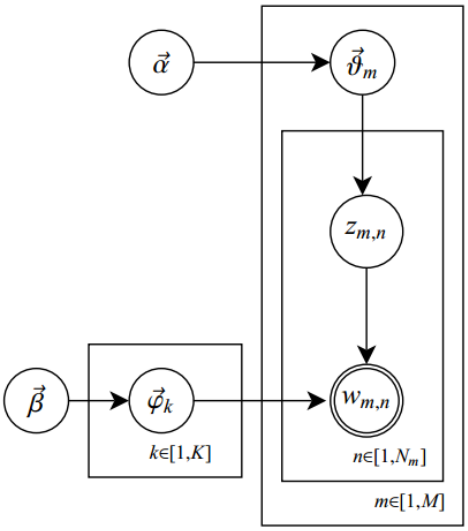


图 1 LDA 模型

**2. 利用 LDA 主题模型进行文本分类**

本文采用以下步骤/思路对金庸的小说集进行文本分类：

1）从给定的 16 本金庸小说数据集中，随机、均匀地抽取$k$个段落，每个段落的标签为对应小说的小说名，每个段落包含$n$个字（$n \geq 500$），每个段落作为一个样本；

2）将随机抽取的$k$个段落（即$k$个样本）中的80%作为训练样本，剩余20%作为测试样本。训练样本数为$k_1 = 80\%k$，测试样本数为$k_2 = 20\%k$；

3）指定主题数为$d$，利用上述$k_1$个训练样本训练 LDA 模型；

4）利用训练好的 LDA 模型得到上述$k_1$个训练样本的主题分布。由于主题数为$d$，因此每个训练样本得到的主题分布为一个$1 \times d$的向量；所有训练样本的主题分布则为一个$k_1 \times d$的特征向量；

5）利用上述训练样本的$k_1 \times d$的特征向量以及对应的$k_1$个标签训练一个线性 SVM 分类器；

6）上述训练样本的$k_1 \times d$的特征向量通过训练好的 SVM 分类器，得到训练样本的预测标签，与真实的标签进行比较，计算训练样本文本分类准确率；

7）利用训练好的 LDA 模型得到$k_2$个测试样本的主题分布。同理，由于主题数为$d$，因此每个测试样本得到的主题分布为一个$1 \times d$的向量；所有测试样本的主题分布则为一个$k_2 \times d$的特征向量；该特征向量通过训练好的 SVM 分类器，得到测试样本的预测标签，与真实的标签进行比较，计算测试样本文本分类准确

率。

其中，上述步骤（1）~（3）为数据准备、预处理和训练 LDA 模型；步骤（4）（5）为训练线性 SVM 分类器；步骤（6）（7）为计算训练和测试样本的文本分类准确率。

## 三、实验结果

本次实验测试了不同的段落（文档）数、每个段落的字数、不同主题数对文本分类准确率的影响。此外，还考察了是否去除停用词对分类准确率的影响。没有什么实际含义的功能词，或用十分广泛但对这样的词搜索引擎无法保证能够给出真正相关的搜索结果、难以帮助缩小搜索范围的词。实验中停用词表由百度、哈工大等创造的停用词表给出。去除停用词有助于数据清洗得更干净、获得的文本更具有实际含义。

实验结果如下表所示。

| 序号 | 主题数 | 段落数 | 每段字数 | 是否去除停用词 | 训练集准确率（%） | 测试集准确率（%） |
|---|---|---|---|---|---|---|
| 1 | 50 | 200 | 500 | 否 | 34.38 | 2.50 |
| 2 | 50 | 200 | 500 | 是 | 41.25 | 12.50 |
| 3 | 50 | 1000 | 500 | 否 | 23.25 | 15.50 |
| 4 | 50 | 1000 | 500 | 是 | 30.63 | 25.00 |
| 5 | 50 | 1000 | 5000 | 否 | 44.00 | 46.00 |
| 6 | 50 | 1000 | 5000 | 是 | 52.75 | 55.00 |

## 四、结果分析

从实验结果可以看出：

① 对比 1 和 2，或 3 和 4，或 5 和 6 的结果，可以看出，去除无意义的停用词，可以增强样本中文本的实际含义，显著增强训练集和测试集的文本分类正确率；

② 对比 1 和 3，或 2 和 4 的结果，可以看出，增加抽取的段落（文档）数，可以显著提高测试集文本分类准确率，但训练集文本分类准确率有所降低，可能是因为训练样本（段落数）太少的时候，训练集上容易引起过拟合导致；

③ 对比 3 和 5，或 4 和 6 的结果，可以看出，增加抽取段落的每段话字数，可以显著增加训练集和测试集上的文本分类准确率；

附录：实验代码

```
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/dxy/codes/NLP_homework/NLP_homework3/stopwords/"):
# replace this path with the stopwords path
        with open("/data1/dxy/codes/NLP_homework/NLP_homework3/stopwords/"+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/dxy/codes/NLP_homework/NLP_homework1/txt_files/' # 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)))
```