机器学习Lab4实验报告

Topic-Model——LDA实验

PB19020499 桂栋南

实验原理

总述

LDA模型是一种基于文本的生成模型，是基于贝叶斯的话题模型。

假设的分布：

- 文本由话题的多项分布表示；
- 话题由单词的多项分布表示；
- 文本的话题分布是狄利克雷分布；
- 话题的单词分布是狄利克雷分布。

LDA是含有因变量的概率图模型：

其中隐变量为：

- 话题的单词分布；
- 文本的话题分布；
- 文本中各个位置的话题。

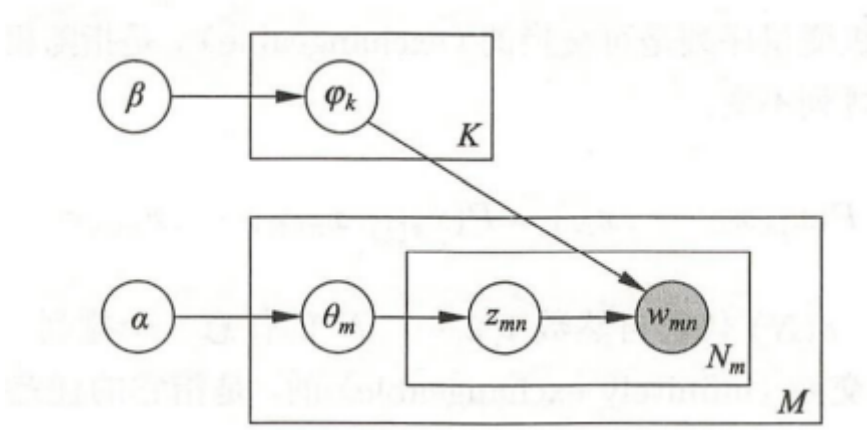
观测变量为：

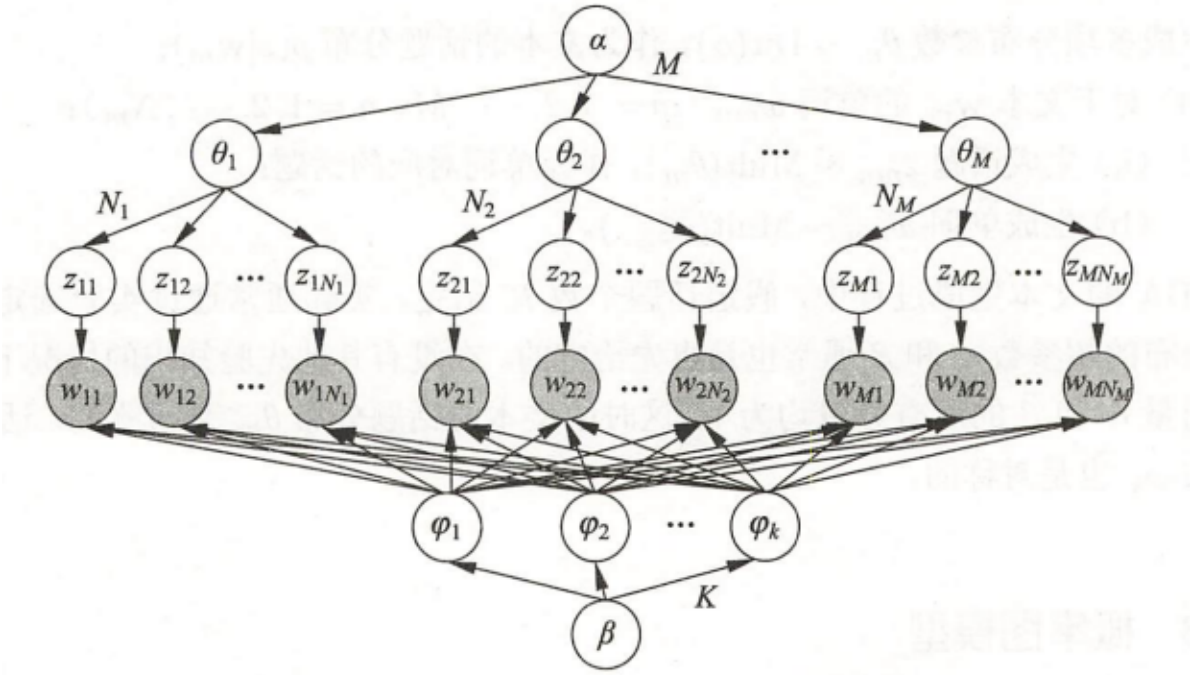
- 文本中各个位置的单词。

模型定义

符号	含义	代码
V	单词的总数	V
$W=\{w_1,\cdots,w_V\}$	文本集合，w_v是第v个单词	
M	文章总数	M
$D=\{\mathbf{w}_1,\cdots,\mathbf{w}_M\}$	文本集合，$\mathbf{w}_m$是第m个文本， $\mathbf{w}_m=(w_{m1},\cdots,w_{mN_m})$	
K	话题总数	K
$Z=\{z_1,\cdots,z_K\}$	话题集合，z_k是第k个话筒	
$\varphi_k=(\varphi_{k1},\cdots,\varphi_{kV})$	φ_{kv}表示话题z_k生成单词w_v的概率$p(w\|z_k)$，服从狄利克雷分布	
$\beta=\{\beta_1,\cdots,\beta_V\}$	φ_{kv}的狄利克雷分布的超参数	beta
$\theta_m=\{\theta_{m1},\cdots,\theta_{mK}\}$	θ_{mk}表示文本$\mathbf{w}_m$生成话题z_k的概率$p(z\|\mathbf{w}_m)$，服从狄利克雷分布	
$\alpha=\{\alpha_1,\cdots,\alpha_K\}$	θ_{mk}的狄利克雷分布的超参数	alpha
n_{mk}	第m篇文档中由k这个topic产生的单词计数	nmk
n_{kv}	第k个topic产生单词v的计数	nkv
n_m	第m篇文档中产生全部单词的计数	nm
n_k	第k个topic产生全部单词的计数	nk

概率图模型





LDA生成过程

1. 根据β生成$\varphi_k \sim \mathrm{Dir}(\beta)$，作为话题$z_k$的单词分布$p(w|z_k)$
2. 根据α生成$\theta_m \sim \mathrm{Dir}(\alpha)$，作为文本$\mathbf{w}_m$的话题分布$p(z|\mathbf{w}_m)$
3. 根据θ_m生成话题$z_{mn} \sim \mathrm{Multi}(\theta_m)$，作为单词$w_{mn}$对应的话题
4. 根据φ_k和z_{mn}生成单词$w_{mn} \sim \mathrm{Multi}(\varphi_{z_{mn}})$，得到结果

LDA算法推导

推断目标：

1. 话题序列的集合$\mathbf{z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_m\}$的后验概率分布
2. 参数$\theta = \{\theta_1, \cdots, \theta_M\}$
3. 参数$\varphi = \{\varphi_1, \cdots, \varphi_K\}$

后验概率$p(\mathbf{z}|\mathbf{w}, \alpha, \beta)$的吉布斯抽样

$$p(\mathbf{z}|\mathbf{w}, \alpha, \beta) \propto p(\mathbf{w}, \mathbf{z}|\alpha, \beta) = p(\mathbf{w}|\mathbf{z}, \beta)p(\mathbf{z}|\alpha)$$

其中有：

$$p(\mathbf{w}|\mathbf{z}, \beta) = \prod_{k=1}^{K}\prod_{v=1}^{V}\varphi_{kv}^{n_{kv}} = \prod_{k=1}^{K}\frac{\mathrm{B}(n_k + \beta)}{\mathrm{B}(\beta)}$$

$$p(\mathbf{z}|\alpha) = \int p(\mathbf{z}|\theta)p(\theta|\alpha)\mathrm{d}\theta = \prod_{m=1}^{M}\frac{\mathrm{B}(n_m + \alpha)}{\mathrm{B}(\alpha)}$$

所以

$$p(\mathbf{z}|\mathbf{w}, \alpha, \beta) \propto \prod_{k=1}^{K}\frac{\mathrm{B}(n_k + \beta)}{\mathrm{B}(\beta)} \cdot \prod_{m=1}^{M}\frac{\mathrm{B}(n_m + \alpha)}{\mathrm{B}(\alpha)}$$

于是满条件分布

$$p(z_i|\mathbf{z}_{-i}, \mathbf{w}, \alpha, \beta) = \frac{1}{Z_{z_i}}p(\mathbf{z}|\mathbf{w}, \alpha, \beta) \propto \frac{n_{kv} + \beta_v}{\sum_{v=1}^{V}(n_{kv} + \beta_v)} \cdot \frac{n_{mk} + \alpha_k}{\sum_{k=1}^{K}(n_{mk} + \alpha_k)}$$

参数估计

$$\theta_{mk}=\frac{n_{mk}+\alpha_k}{\sum_{k=1}^{K}\left(n_{mk}+\alpha_k\right)}$$

$$\varphi_{kv}=\frac{n_{kv}+\beta_v}{\sum_{v=1}^{V}\left(n_{kv}+\beta_v\right)}$$

LDA-Gibbs算法

输入: 文本的单词序列 $\mathbf{w}=\{\mathbf{w}_1,\cdots,\mathbf{w}_m,\cdots,\mathbf{w}_M\},\mathbf{w}_m=(w_{m1},\cdots,w_{mn},\cdots,w_{m_{N_m}})$;
输出: 文本的话题序列 $\mathbf{z}=\{\mathbf{z}_1,\cdots,\mathbf{z}_m,\cdots,\mathbf{z}_M\},\mathbf{z}_m=\left(z_{m1},\cdots,z_{mn},\cdots,z_{m_{N_m}}\right)$ 的后验概率分布 $p(\mathbf{z}\mid\mathbf{w},\alpha,\beta)$的样本计数, 模型的参数 φ和θ 的估计值;
参数: 超参数 α 和 β , 话题个数 K。

1. 设所有计数矩阵的元素n_{mk},n_{kv}, 计数向量的元素n_m,n_k 初值为 0 ;
2. 对所有文本 $\mathbf{w}_m,m=1,2,\cdots,M$
   对第 m 个文本中的所有单词 $w_{mn},n=1,2,\cdots,N_m$
   1. 抽样话题 $z_{mn}=z_k\sim\text{Mult}\left(\frac{1}{K}\right)$;
   2. 增加文本-话题计数 $n_{mk}=n_{mk}+1$,增加文本-话题和计数 $n_m=n_m+1$,增加话题-单词计数 $n_{kv}=n_{kv}+1$,增加话题-单词和计数 $n_k=n_k+1$;
3. 循环执行以下操作, 直到进入燃烧期
   对所有文本 $\mathbf{w}_m,m=1,2,\cdots,M$
   对第 m 个文本中的所有单词 $w_{mn},n=1,2,\cdots,N_m$
   1. 当前的单词 w_{mn} 是第 v个单词, 话题指派 z_{mn}是第 k 个话题;
   2. 减少计数 $n_{mk}=n_{mk}-1,n_m=n_m-1,n_{kv}=n_{kv}-1,n_k=n_k-1$;
   3. 按照满条件分布进行抽样

      $$p\left(z_i\mid\mathbf{z}_{-i},\mathbf{w},\alpha,\beta\right)\propto\frac{n_{kv}+\beta_v}{\sum_{v=1}^{V}\left(n_{kv}+\beta_v\right)}\cdot\frac{n_{mk}+\alpha_k}{\sum_{k=1}^{K}\left(n_{mk}+\alpha_k\right)}$$

      得到新的第 k'个话题, 分配给 z_{mn} ;
   4. 增加计数 $n_{mk'}=n_{mk'}+1,n_m=n_m+1,n_{k'v}=n_{k'v}+1,n_{k'}=n_{k'}+1$;
   5. 得到更新的两个计数矩阵 $N_{K\times V}=[n_{kv}]$ 和 $N_{M\times K}=[n_{mk}]$, 表示后验概率分布 $p(\mathbf{z}\mid\mathbf{w},\alpha,\beta)$ 的样本计数;利
4. 用得到的样本计数, 计算模型参数

$$\theta_{mk}=\frac{n_{mk}+\alpha_k}{\sum_{k=1}^{K}\left(n_{mk}+\alpha_k\right)}$$

$$\varphi_{kv}=\frac{n_{kv}+\beta_v}{\sum_{v=1}^{V}\left(n_{kv}+\beta_v\right)}$$

实验实现

总体框架

```
class LDA():

    def __init__(self, alpha=6, beta=0.01, iter=50, topics_num=8):
        self.alpha = alpha
        self.beta = beta
        # 最大迭代次数
```

```
        self.iter = iter
        # 所有doc的列表和停用词列表
        documents, stopwords = self.read_file()
        print("Reading file has finished!")
        # docs是二维列表，docs[i][j]是第i篇文章第j个单词的对应的编号
        # word2id是字典，记录关键字是单词，值为对应编号的转换关系
        # id2word是字典，记录关键字是编号，值为对应单词的转换关系
        self.docs, self.word2id, self.id2word = self.preprocessing(
            documents, stopwords)
        print("Preprocessing has finished!")
        # M是文章数量
        self.M = len(self.docs)
        # V是所有单词的数量
        self.V = len(self.word2id)
        # Z是二维列表，z[i][j]记录第i篇文章第j个单词的topic
        self.Z = []
        # K是topic数量
        self.K = topics_num
        # 初始化
        # nmk是array(M*K),nmk[m][k]表明第m篇文档中由k这个topic产生的单词计数
        self.nmk = np.zeros([self.M, self.K]) + self.alpha
        # nkv是array(K*V),nkv[k][v]表明第k个topic产生单词v的计数
        self.nkv = np.zeros([self.K, self.V]) + self.beta
        # nk是array(K),nk[k]表明第k个topic产生全部单词的计数
        self.nk = np.zeros([self.K]) + self.V * self.beta

    # 读取文件
    def read_file(self):
        pass

    # 预处理(分词，去停用词，为每个word赋予一个编号，文档使用word编号的列表表示)
    def preprocessing(self,documents, stopwords):
        pass

    # 进行初始化的多项分布的分配
    def multinomial(self):
        pass

    # gibbs采样
    def gibbsSampling(self):
        pass

    def perplexity(self):
        pass

    def run(self):
        pass

```

- `init`初始化读文件并进行预处理，定义模型参数。
- `run`函数调用初始化的`multinomial`函数对话题采样，并调用进行循环的`gibbsSampling`函数进行话题采样，并调用`perplexity`函数存储生成匹配情况。

预处理函数

```
# 预处理(分词，去停用词，为每个word赋予一个编号，文档使用word编号的列表表示)
    def preprocessing(self,documents, stopwords):
        word2id = {}
        id2word = {}
        docs = []
        currentDocument = []
        currentWordId = 0

        flag_list = ['n','nz','vn']
        for document in documents:
            # 分词
            segList = psg.cut(document)
            for seg_word in segList:
                word = seg_word.word
                # 单词长度大于1，并且不包含数字和单词，并且不是停止词，并且词性符合要求
                if len(word) > 1 and not re.search('[0-9]', word) \
                    and word not in stopwords and not re.search('[a-z]',
word) and not re.search('[A-Z]', word)\
                        and seg_word.flag in flag_list:
                    # word出现过，docs直接增加其编号
                    if word in word2id:
                        currentDocument.append(word2id[word])
                    else:
                        # 增加一个新的word-ID，加入docs
                        currentDocument.append(currentWordId)
                        # 记录word->id的对应关系
                        word2id[word] = currentWordId
                        # 记录id->word的对应关系
                        id2word[currentWordId] = word
                        # 准备下一个word-ID
                        currentWordId += 1
            docs.append(currentDocument)
            currentDocument = []
        return docs, word2id, id2word

```

- 该函数实现预处理功能。
    - 实现去除停用词和单词数字；
    - 记录单词word和训练的id之间的匹配关系。

初始多项分布的话题分配

```
def multinomial(self):
    for d, doc in enumerate(self.docs):
        # d是文档的序号，doc是文档内容相应单词的编号
        zCurrentDoc = []
        for w in doc:
            z = np.random.multinomial(1, [1/self.K]*self.K).argmax()
            zCurrentDoc.append(z)
            self.nmk[d, z] += 1
            self.nkv[z, w] += 1
            self.nk[z] += 1
            self.nm[d] += 1
        self.z.append(zCurrentDoc)
```

- 该函数实现初始化的多项分布的分配。

Gibbs采样

```
# gibbs采样
    def gibbsSampling(self):
        # 为每个文档中的每个单词重新采样topic
        for d, doc in enumerate(self.docs):
            # d是文档的序号，doc是文档内容相应单词的编号
            for index, w in enumerate(doc):
                # index是单词w在该篇文章doc中的序号位置
                z = self.z[d][index]
                # 将当前文档当前单词原topic相关计数减1
                self.nmk[d, z] -= 1
                self.nkv[z, w] -= 1
                self.nk[z] -= 1
                self.nm[d] -= 1
                # 重新计算当前文档当前单词属于每个topic的概率
                pz = np.multiply((self.nkv[:,
w]+self.beta)/np.sum(self.nkv[:, w]+self.beta),
                                 (self.nmk[d,
:]+self.alpha)/np.sum(self.nmk[d, :]+self.alpha))
                # 按照计算出的分布进行采样
                z = np.random.multinomial(1, pz / pz.sum()).argmax()
                self.z[d][index] = z
                # 将当前文档当前单词新采样的topic相关计数加上1
                self.nmk[d, z] += 1
                self.nkv[z, w] += 1
                self.nk[z] += 1
                self.nm[d] += 1
```

- 该函数实现吉布斯采样过程。
    - 每次采样是对于每篇文档的每个单词的主题的重新分配，分配基于推导得到的参数的多项分布。

`run`函数

```
def run(self):
    self.multinomial()
    # 记录匹配度
    self.perplexity_score = []
    for i in range(0, self.iter):
```

```
        self.gibbsSampling()
        self.perplexity_score.append(self.perplexity())
        # 打印iter信息
        print(time.strftime('%X'), "Iteration: ", i, " Completed",
              " Perplexity: ", self.perplexity_score[-1])
        if i < 10:
            continue
        # 如果匹配度大于前十次的平均值，停止迭代
        if self.perplexity_score[-1] >
np.average(self.perplexity_score[-10:]):
            break
```

- 该函数调用`multinomial`和`gibbsSampling`进行LDA算法的主要功能
  - `gibbsSampling`的停止条件是perplexity大于前十次的平均值，表明该值不在下降，迭代停止。

`get_topic`函数

```
def get_topic(self, maxTopicWordsNum=10):
    topicwords = []
    for z in range(0, self.K):
        # 对于每个主题，对在该主题下出现单词的数量排序
        ids = self.nkv[z, :].argsort()
        topicword = []
        # 根据id->word的转换关系转为id
        for j in ids:
            topicword.insert(0, self.id2word[j])
            # 返回需求的每个主题的相应数量的单词
            topicwords.append(topicword[0: maxTopicWordsNum])
    return topicwords
```

- 该函数返回需要的每个主题的前maxTopicWordsNum个主题单词。

实验结果

主题情况

```
array([['研究', '科学家', '人类', '时间', '人员', '流感', '技术', '消息', '科技', '利用',
        '太空', '航天飞机', '流感病毒', '海豚', '科学'],
       ['学生', '经济', '大学', '学校', '工作', '政府', '计划', '教育', '专业', '记者', '情况',
        '银行', '孩子', '能力', '国家'],
       ['网站', '手机', '互联网', '公司', '网络', '信息', '服务', '网游', '用户', '业务',
        '视频', '内容', '彩票', '记者', '全国'],
       ['专家', '网友', '老师', '压力', '分析', '走势', '突破', '黄金', '成本', '股票', '新浪',
        '大盘', '建议', '调整', '机会'],
       ['比赛', '主队', '赔率', '球队', '主场', '火箭', '客场', '数据', '球员', '公司', '联赛',
        '时间', '情况', '篮板', '奇才'],
       ['游戏', '比赛', '电子竞技', '玩家', '海选', '总决赛', '冠军', '赛事', '战队', '赛区',
        '世界', '项目', '奖金', '时间', '星际争霸'],
       ['项目', '市场', '发展', '投资', '生活', '建筑', '企业', '活动', '地产', '国际', '新浪',
        '文化', '空间', '产品', '设计'],
       ['电影', '主持人', '票房', '影片', '观众', '演员', '作品', '朋友', '故事', '娱乐',
        '现场', '新浪', '感觉', '电影节', '角色']], dtype='<U4')
```

科技
教育
彩票
股票
体育
游戏
房产
娱乐

- 红字是根据`data.xlsx`得到的主题对应情况。

可视化

简单根据gibbs采样50次的运行结果作出下图：