实验三 文本处理与分析

【实验目的】

1. 掌握文本预处理的基本技术
2. 掌握倒排文档的实现
3. 掌握向量空间模型的实现
4. 掌握文本分类模型的实现

【实验环境】

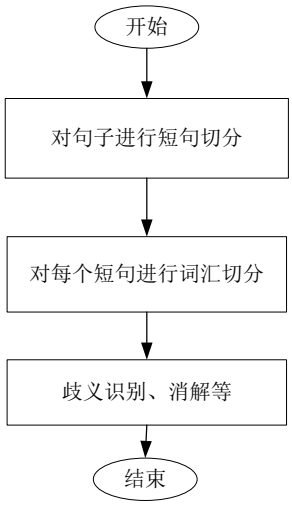
1. Python 3
2. Windows/Linux 操作系统

【实验内容】

一、 文本预处理

1. 词汇切分

词汇切分，指的就是将一个完整的句子划分为一个个词条的过程。分词是文本处理挖掘的基础，也便于搜索引擎建立词条索引。句子词汇切分的流程包含三大步骤，如下图所示。



第一，对输入的文档（主要面向中文文档）进行预处理，得到单个中文短句的集合。通过标点符合（逗号、句号等）进行切分，缩小中文分词的句子长度；

第二，对每个短句进行词汇切分，即运用一定的分词算法将其中的词汇切分开来；

第三，进行分词结果的优化，主要是考虑到分词中可能存在的错误，对错误进行识别和纠正。该步骤的主要任务之一就是歧义的识别和消解。

现有的分词算法可分为三大类：基于字符串匹配的分词方法、基于理解的分词方法和基于统计的分词方法。

- 基于字符串匹配的分词

该方法也称为基于词典的分词方法，即通过设定词典，按照一定的字符串匹配方法把存在于词典中的词从句子中切分出来。该方法有三个基本要素：**分词词典**、**文本扫描顺序**（正向扫描、逆向扫描和双向扫描）和**匹配原则**（最大匹配、最小匹配、逐词匹配和最佳匹配）。

本实验主要考察正向最大匹配法。所谓最大匹配，就是优先匹配最长词汇，即每一句的分词结果中的词汇总量要最少。正向最大匹配分词在实现上可以采用减字法，首先需要将词典中词汇按照其长度从大到小的顺序排列，然后对于待切分的中文字符串，做如下处理：

（1）将字符串和词典中的每个词汇逐一进行比较；

（2）如果匹配到，则切分出一个词汇，转步骤 5 执行；

（3）否则，从字符串的末尾减去一个字；

（4）如果剩下的字符串只有一个字，则切分出该字；

（5）将剩下的字符串作为新的字符串，转步骤 1 执行，直到剩下的字符串长度为 0。

实验要求：按照上述逻辑实现正向减字最大匹配法，以字符串“今天是中华人民共和国获得奥运会举办权的日子”为测试用例，代码使用的词典可参考使用dict_example.txt。

示例：

[1] s=“今天是中华人民共和国获得奥运会举办权的日子”

[2] s=“今天是中华人民共和国获得奥运会举办权的日”

[3]s=“今天是中华人民共和国获得奥运会举办权的”

…

[20]s=“今天”

[21]s=“是中华人民共和国获得奥运会举办权的日子”

[22]s=“是中华人民共和国获得奥运会举办权的日”

…

[39]s=“是”

[40]s=“中华人民共和国获得奥运会举办权的日子”

…

[51]s=“中华人民共和国”

[52]s=“获得奥运会举办权的日子”

…

[61]s=“获得”

思考：词条一般都有一定的长度，我们假定最长词包含的汉字个数 len = 7，那么每次可以取 len 长度的字符串和字典进行比较。实现上述改进的正向减字最大匹配法。

示例：

[1] s=“今天是中华人民”

…

[6] s=“今天”

[7]s=“是中华人民共和”

[13] s=“是”

[14]s=“中华人民共和国”

[15]s=“获得奥运会举办”

[20] s=“获得”

[21] s=“奥运会举办权的”

…

2. 去停用词

停用词在不同的文本分析任务中有着不同的定义，在基于词的检索系统中，停用词是指出现频率太高、没有太大检索意义的词，如“一个、一种、因此、否则、其中”等；在文本分类中，停用词是指没有意义的虚词和类别色彩不强的中性词；在自动问答系统中，停用词因问题不同而动态变化。

实际任务中所采用的停用词表一般为通用停用词表或在其基础上进行增减的停用词表。哈工大停用词表为通用停用词表，其停用词比较全面；百度停用词表为专门进行信息检索的停用词表（“https://github.com/goto456/stopwords”中可以下载）。外文的停用词表，可以到“https://www.ranks.nl/stopwords”网站取查找，包含了多种余元的停用词表。

3. Python 开源库 jieba 的使用

“结巴”（jieba）分词是 Python 的一个中文分词组件，是基于 Trie 树结构实现高速的词图扫描，生成句子中汉字所有可能成词情况构成的有向无环图，然后采用动态规划查找最大概率路径，找出基于词频的最大切分组合。它支持中文分词、用户自定义词典、关键词提取、词性标注和并行分词等。

- jieba.cut(sentence, cut_all = False, HMM = True)

  该函数有三个参数，sentence 是需要分词的字符串，其编码可以是 unicode、utf-8 或 gbk；cut_all 用来控制是否采用全模式，默认为否；用来控制是否使用 HMM 模型，默认值为 True。该方法的返回结果是一个 generator 类型的对象，可以通过转换为 list 或者使用 for 循环获取切分结果。

安装工具包 pip install jieba

示例代码：基于 jieba 的分词

```
import jieba

# 精简模式
ds = jieba.cut("今天是中华人民共和国获得奥运会举办权的日子.")
print(list(ds))

# 全模式
ds = jieba.cut("今天是中华人民共和国获得奥运会举办权的日
子.",cut_all=True)
print(list(ds))
```

- 加载自定义词典

文本处理中经常会遇到新词、专业词汇，例如“大数据”会被切分为“大”和“数据”两个词。这个问题就可以通过加载自定义词典来解决。具体方法如下：

先创建一个文本文件，其中每行编写一个词汇，并保存文件，例如保存在“E:\udict.txt”中。然后通过 load_userdict 方法加载，即可进行切分。

示例代码：加载自定义词典的分词

```
import jieba
# 首先将 大数据 保存在自定义字典中
jieba.load_userdict(r"E:\udict.txt")
ds = jieba.cut("这是一本大数据相关专业的教材.")
print(list(ds))
```

二、 倒排文档的实现

倒排文档是一种面向单词的索引机制，是将顺排文档中的词汇取出，按一定规则排序，归并相同词汇，并把在顺排文档中相关记录的记录号集合赋予其后，以保证通过某一特征词能够快速、方便地获取相关记录。

示例代码：倒排文档的构建和查询

```
import jieba

def mapper(lineNum, list):
    dic = {}
    for item in list:
        key = ''.join([str(lineNum), ':', item])
        if key in dic:
            ll = dic.get(key)
            ll.append(1)
            dic[key] = ll
        else:
            dic[key] = [1]
    return dic

def reducer(dic):
    keys = dic.keys()
```

```
    rdic = {}
    for key in keys:
        lineNum, kk = key.split(":")
        ss = ''.join([lineNum, ':', str(dic.get(key))])
        if kk in rdic:
            ll = rdic[kk]
            ll.append(ss)
            rdic[kk] = ll
        else:
            rdic[kk] = [ss]
    return rdic

def combiner(dic):
    keys = dic.keys()
    tdic = {}
    for key in keys:
        valuelist = dic.get(key)
        count = 0
        for i in valuelist:
            count += i
        tdic[key] = count
    return tdic

def get_reverse_index(filepath):
    file = open(filepath, 'r', encoding="utf8")
    lineNum = 0
    rdic_p = {}
    while True:
        lineNum += 1
        line = file.readline()
        if line != '':
            print(lineNum, ' ', line)
        else:
            break
        # 先分词
```

```
            word_list = list(jieba.cut(line))
            mdic = mapper(lineNum, word_list)
            cdic = combiner(mdic)
            print(cdic)
            rdic_p.update(cdic)

        rdic = reducer(rdic_p)
        print(rdic)
        return rdic

if __name__ == '__main__':
        # data  文档
        dic = get_reverse_index('data.txt')
        while(1):
            search_word = input('Please input the word you want to search: ')
            if (search_word in dic):
                print(dic.get(search_word))
            else:
                print(-1)
```

思考：添加去停用词的实现，重新输出结果。

三、 向量空间模型的实现

文本是信息检索系统处理的主要数据类型，是一种典型的非结构化数据。在获得文本中的词汇及其特征之后，需要一种合适的模型对这些特征进行数学表示，以便基于文本内容的各种分析应用能够有效地展开。向量空间模型是当前最常用的模型之一，使用特征向量来表示文档，本实验主要介绍向量空间模型的实现。

文本的向量空间模型和线性代数中学过的向量空间模型是相同的，由基向量和坐标构成。以词汇作为维度为例，在文本表示中，基向量就是特征词汇，坐标就是词汇的权重。

通常情况下，文本的词汇量会很大，直接选取分词后得到的词作为文本的特征项时不可取的（维度很高），因此需要进行特征项的选择。假定选出来的词汇集合 $W=\{w_1, w_2, \ldots, w_n\}$，则表示文本有 n 个特征词汇。

- 基向量是：
  - $\overrightarrow{w1} = (w_1, 0, 0, \ldots, 0)$
  - $\overrightarrow{w2} = (0, w_2, 0, \ldots, 0)$
  - ...
  - $\overrightarrow{wn} = (0, 0, 0, \ldots, w_n)$
- 这样对于任意一个文本，就可以表示成为：
  $\vec{d} = x_1\overrightarrow{w1} + x_2\overrightarrow{w2} + \cdots + x_n\overrightarrow{wn}$

其中，x_1、x_2、…、x_n就是文本在每个基向量上的权重。通过这种方式，就可以把一个文本表示为一个特征向量。权重的计算方式有布尔权重（出现、未出现）、词频（特征项频率权重）和 TF-IDF（词频率-逆文档频率）。

$$TF\text{-}IDF(a_j, t_i) = \frac{n_{i,j}}{\sum_{i=1}^{n} n_{i,j}} * \log\frac{N}{N(t_i)}$$

为避免 IDF 为 0，即 $N=N(t_i)$，可以平滑处理 $IDF=\log(1+N/N(t_i))$。

- 使用 Python 构建向量空间表示的基本步骤
  #装载停用词列表
  #加载自定义词典
  #分词、去停用词
  #特征词项选择
  #使用 TfidfVectorizer 计算每个文档中每个词汇的 TF-IDF 值

安装工具包：pip install sklearn　pip install gensim
示例代码：构建向量空间表示

```
import jieba
from gensim.corpora.dictionary import Dictionary
from sklearn.feature_extraction.text import TfidfVectorizer
#载入四篇测试文档
docs=["新型互联网大数据技术研究",
      "大数据采集技术与应用方法",
      "一种互联网技术研究方法",
      "计算机系统的分析与设计技术"
      ]

#装载停用词列表
stoplist=open('stopword.txt','r',encoding="utf-8").readlines()
```

```
stoplist = set(w.strip() for w in stoplist)

#加载自定义词典，添加“大数据”词汇
jieba.load_userdict("userdict.txt")

#分词、去停用词
texts=[]
for d in docs:
    doc=[]
    for w in list(jieba.cut(d,cut_all=False)):
        if len(w)>1 and w not in stoplist:
            doc.append(w)
    texts.append(doc)

#特征选择，假设依据是：词汇至少出现 2 次，而且词汇所在的段落数/总的段落数<=1.0
#选择符合这两个条件的前 10 个词汇作为页面内容的代表
dictionary = Dictionary(texts)
dictionary.filter_extremes(no_below=2, no_above=1.0, keep_n=10)
d=dict(dictionary.items())
docwords=set(d.values())
print("维度词汇是：",docwords)

#将 texts 中的每个文档按照选择出来的维度词汇重新表示文档集,并转换成为 TfidfVectorizer 的格式
corpus=[]
for text in texts:
    d=""
    for w in text:
        if w in docwords:
            d=d+w+" "
    corpus.append(d)

print(corpus)
```

```
# 计算每个文档中每个词汇的 tf-idf 值
vectorizer = TfidfVectorizer()
tfidf = vectorizer.fit_transform(corpus)

words = vectorizer.get_feature_names()
for i in range(len(corpus)):
    print('----Document %d----' % (i))
    for j in range(len(words)):
        print( words[j], tfidf[i,j])
```

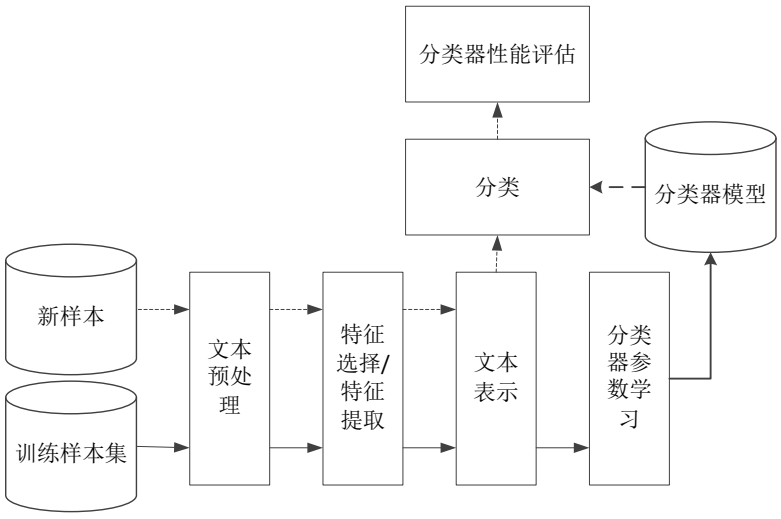
四、 文本分类的实现

文本分类是指根据文本内容和形式的异同，按照一定的体系有系统地组织和区分文本。文本分类有两种方法，即基于规则的方法和基于机器学习的方法。基于规则的方法是采用语言学的知识对文本进行分析后，人工定义一系列启发式规则，用于文本分类；基于机器学习的方法是对文本进行特征提取，然后采用分类器进行分类。目前主流的研究方法就是采用机器学习的方法。

在分类中涉及到的概念有：分类器、训练、训练样本、测试样本等。

- 分类器是对数据挖掘中对样本进行分类的总称，
- 训练是指对模型的参数进行优化，选取最优的模型参数使得算法能够建立具有很好泛化能力的模型。
- 训练样本是由类别已知的样本组成，用于模型的训练。
- 测试样本是由类别未知的样本组成，用于测试模型的性能。

一个文本分类的基本流程如下图所示，包含了分类器训练（实线）和样本分类（虚线）两个过程。



- 文本分类模型

常用的文本分类器有朴素贝叶斯分类、KNN 分类、SVM 分类和神经网络分类等。本实验主要介绍支持向量机（SVM）的实现示例。支持向量机（SVM）分类算法基本思想是使用简单的线形分类器划分样本空间。对于在当前特征空间中线性不可分的模式，则使用一个核函数把样本映射到一个高维空间中，使得样本能够线性可分。常用的核函数有线性核函数、多项式核函数、径向基核函数、Sigmoid 核函数和复合核函数。

SVM 基于结构风险最小化原则。它在向量空间中找到一个决策面，这个面能“最好”地分割两个分类中的数据点。两个分类的分类间隔可以定义“最好”分割。线性可分情况下的决策面是平面，SVM 就是要在训练集中找到具有最大分类间隔的决策平面。

- 分类模型的性能评估

	分类为正例	分类为负例
实际为正例	TP	FN
实际为负例	FP	TN

分类查准率：查准率（Precision，记为 *P*），指被正确分类的正例占所有被分为正例的数据的比值。即：

$$P = \frac{TP}{TP + FP}$$

分类查全率：查全率（Recall，记为 *r*），指被正确分类的正例占测试集中正例数的比。即：

$$R = \frac{TP}{TP + FN}$$

综合指标 F1：用以反映查准率 *P* 和查全率 *R* 的综合效果。公式：

$$F_1 = \frac{2PR}{P + R}$$

正确率：

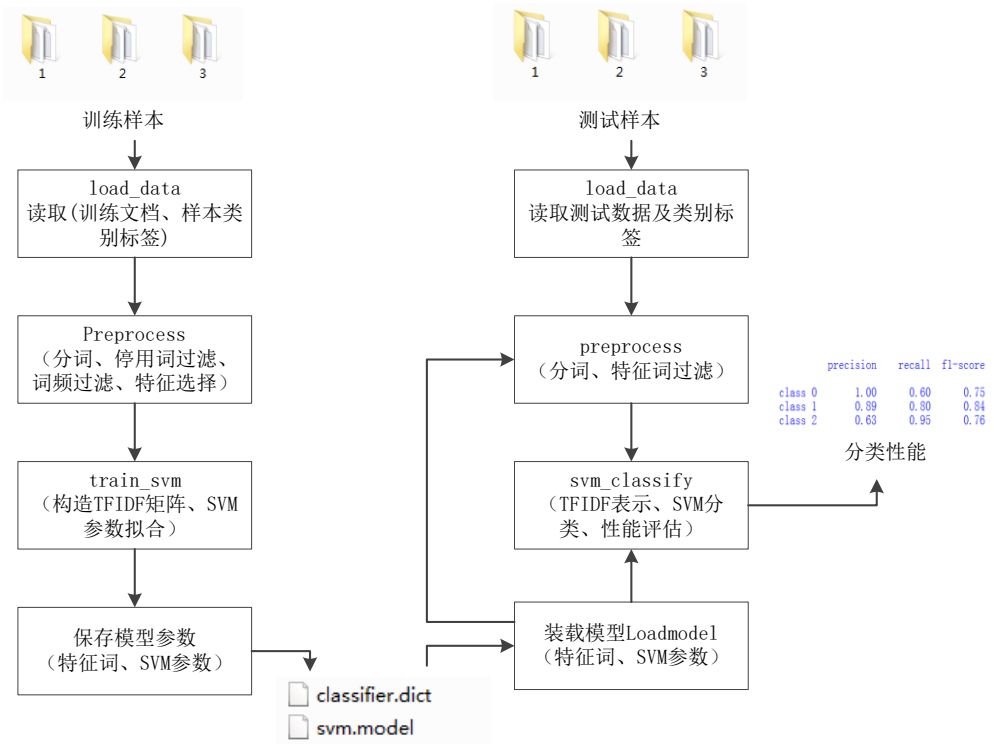
$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

此外，还有宏平均和微平均的评价指标，请同学们自行学习。

- 新闻文本分类的示例

本实验介绍基于 SVM 分类器进行新闻文本分类的 Python 实现方法，总体流程如下图所示。数据集中包含 3 个新闻类别，分别是汽车类、财经类、科技类。

训练集中每个类别有 130 个训练样本，测试集中每个类别有 20 个样本。每个样本都单独保存到一个文本文件中。



示例代码：见搜索引擎实验 2 代码。首先运行 train_classifier 进行模型训练，将分类器的参数和特征词保存起来；

```
请输入训练集的的根目录：data\train
```

然后运行 classify 进行测试，并输出分类器的分类性能。需要说明的是，在分类（测试）中输入包含训练集的根目录，是为了在整个文本集中计算词汇的 IDF 值。

```
请输入模型文件的目录：.
请输入包含训练集的根目录：data\train
请输入包含测试集的根目录：data\test
```

思考：所谓“近朱者赤”，通过文本的相似度，找到与新文本相似的文本（即近邻）所属的类别，然后猜想新文本也属于这个类别。对于一个测试文本，计算它与训练样本集中每个文本的相似度，找出 K 个最相似的文本，根据加权距离和判断测试文本所属的类别。这种方法叫最近邻法（*k* Nearest Neighbor，kNN）。

实验要求用 kNN 分类器替换上述示例的 SVM 分类器，并比较两种分类器的性能评估结果。可以参考 sklearn 的 kNN 实现（https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification）

实验四 链接分析

【实验目的】

1. 掌握 PageRank 算法的原理和实现

【实验环境】

1. Python 3

2. Windows/Linux 操作系统

【实验内容】

一、 基于 PageRank 的链接分析

链接分析是网络搜索引擎中的一项重要技术。在网页搜索排序时，同时使用基于链接重要性的排序与基于内容相关性的排序可以有效地改善页面的排序结果。

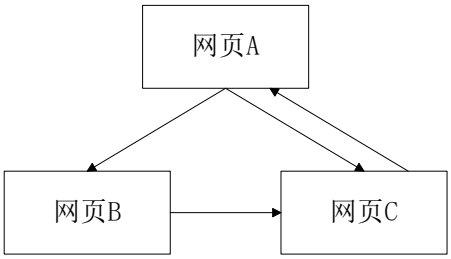
PageRank 算法是一个经典算法，来自于 google 搜索引擎，是一种根据网页之间相互的超链接计算页面级别的方法。它由 Larry Page 和 Sergey Brin 在 20 世纪 90 年代后期发明。由于它解决了网络图中每个节点重要性的量化计算方法，因此在许多可以抽象为网络连接图的应用中得到广泛采用。如计算 Web 页面的重要性、社交网络中的重要人物识别以及文本中的关键词提取等。

一般地，PageRank 计算公式如下：

$$PR_n(A) = (1-d)/N + d \times (\sum_{i=1}^{m} \frac{PR_{n-1}(T_i)}{C(T_i)})$$

其中：$PR_n(A)$是网页 A 的 PageRank 值，PR_{n-1}（T_i）是指网页 T_i 存在指向 A 的链接，并且网页在上一次迭代时的 PageRank 值，C（T_i）是指网页 T_i 的外链数量。d 是平滑因子，N 是页面总数。

实验要求：存在如图所示简单的网页链接关系，页面数 N=3，假定平滑引子 d=0.5。用程序实现迭代计算，输出每个网页 PageRank 值。



最后网页【A, B, C】的 PageRank 值：

```
final result: [0.35897436 0.25641026 0.38461539]
```

实验报告格式，以学号+班级+姓名命名。

【实验目的】

【实验内容】

【实验心得】