《数字媒体技术基础》第二次作业

作业题目：

从以下两个题目中任选一个，完成报告、编程和实验分析:

1. 利用图像直方图特征（如颜色直方图、纹理直方图等） 或者 bag of words 的特征(如局部描述子 SIFT 聚类后的统计信息)计算不少于 10 组图像对的相似度。
2. 利用 bag of words（如词或者字的统计直方图信息） 或者 skip gram（如基于 skip gram 的表示向量） 计算不少于 10 组句子对的相似度。

作业要求

1. 报告部分要求包括所使用技术的描述（占 20%分）
2. 核心算法代码分析（占 20%分）
3. 实验结果分析（占 20%分，不少于 10 对样本的结果分析）
4. 报告书写要求表述严谨、图文并茂、格式规整（占 10%分） 。
5. 要求编写出图形界面，建议基于 python语言编写。其中界面要求可以选择图像对或者输入句子对，并输出相似度值。

实验内容：

1. 实现功能

利用 bag of words计算任意输入句子对的相似度，其中实验测试的样例不少于 10 组，程序实验了图形化的界面，当输入一对句子的指定的空白框中，界面会输出这对句子对的相似度值。

2. 所使用到的技术以及相关知识点：

2.1 词袋模型（Bag of Words）：

所述袋的字模型是在所使用的简化表示自然语言处理及信息检索。在此模型中，文本（例如句子或文档）表示为其单词的包（多集），而忽略了语法甚至单词顺序，但保持了多重性，是一种流行并且简单的特征提取方法。

举个例子：

- 以下面两个句子为例：

```
A likes to play games. B likes too.

B also likes to play games too.
```

- 根据语料中出现的句子分词，构建词袋（每一个出现的词都加进来）.

```
'A', 'likes', 'to', 'play', 'games', '.', 'B', 'likes', 'too', '.', 'B',
'also', 'likes', 'to', 'play', 'games', 'too', '.'
```

- 给每个不同词一个位置索引（需要先构造一个不重复元素的列表）

  不重复列表：

```
{'also', 'play', 'to', '.', 'B', 'A', 'likes', 'too', 'games'}
```

给定位置引索：

```
{'also': 0, 'play': 1, 'to': 2, '.': 3, 'B': 4, 'A': 5, 'likes': 6, 'too':
7, 'games': 8}
```

- 将每个词袋表示为JSON对象，其中元素的顺序是自由的

```
{'also': 0, 'play': 1, 'to': 1, '.': 2, 'B': 1, 'A': 1, 'likes': 2, 'too':
1, 'games': 1}

{'also': 1, 'play': 1, 'to': 1, '.': 1, 'B': 1, 'A': 0, 'likes': 1, 'too':
1, 'games': 1}
```

- 每个文本就可以使用9维的向量来表示

```
[0, 1, 1, 2, 1, 1, 2, 1, 1]
[1, 1, 1, 1, 1, 0, 1, 1, 1]
```

- 这样我们就成功使用词袋模型来提取句子的特征

2.2 余弦相似度（cosine similarity）：

余弦相似度是内积空间的两个非零向量之间相似度的度量。定义为等于它们之间夹角的余弦。

给定A和B的两个向量，余弦相似度cos（θ）使用点积和幅值表示：

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

相似性范围从-1表示完全相反，到1表示完全相同，其中0表示正交或去相关，而中间值表示中间相似或不相似。

对于文本匹配，属性向量A和B通常是文档的术语频率向量。余弦相似度可以看作是在比较过程中标准化文档长度的一种方法。

在信息检索的情况下，两个文档的余弦相似度范围从0到1，因为频率项不能为负。

2.3 Python-Tkinter 图形化界面设计

Tkinter：Python 与Tk GUI工具包的绑定。它是Tk GUI工具包的标准Python接口[1]，并且是Python的事实上的标准GUI。

- 其图像化编程的基本步骤通常包括：
  - 导入tkinter模块 from tkinter import *
  - 创建 GUI 根窗体
  - 添加界面元素，编写相应的函数。
  - mainloop()启动应用程序的主循环，等待鼠标和键盘事件
- 程序使用到的相关函数：
  1. Tk()：初始化一个根窗体实例并返回
  2. title()： 设置窗体的标题文字

3. geometry()： 以像素为单位设置窗体的大小
4. mainloop()：启动应用程序的主循环，等待鼠标和键盘事件
5. place()：例在父容器中的绝对或相对位置参数（相对于根窗体宽和高的比例位置relx,rely和控件实例相对于根窗体的高度和宽度比例relheight,relwidth）进行布局
6. Entry(): 接收单行文本输入的控件,可以对其中的文本进行函数操作：如取其中的文本get(),删除其中的文本delete(起始位置，终止位置)
7. label(): 显示标签，可以设置其中的文本内容text，背景色bg，加粗bd等
8. Text(): 设置文本框
9. Button(): 鼠标单击事件触发运行程序，直接调用函数。参数表达式为“`command=函数名`”，匿名函数调用函数和传递参数。参数的表达式为“`command=lambda`”:`函数名（参数列表）`

- 核心小部件：容器：框架，标签框架，顶层，窗格窗口。按钮：按钮，单选按钮，复选按钮（复选框）和菜单按钮。
- 文本小部件：标签，消息，文本。
- 条目小部件：比例尺，滚动条，列表框，滑块，旋转框，条目（单行），选项菜单，文本（多行）和画布（矢量和像素图形）。
- StringVar是Tkinter下的对象，作用是跟踪变量的值的变化，以保证值的变更随时可以显示在界面上，set可以设置显示在文本框的内容，get函数可以获得文本款的内容

3. 核心算法代码分析

3.1 计算句子对的相似度：

利用Bag of Words将句子提取成向量的形式，通过计算余弦相似度来计算句子对的相似度：

- 第一步是实现输入句子对的分词，核心代码以及详细注释如下：

```
    #导入nltk中的word_tokenize函数
    from nltk import word_tokenize

    sentences = [sentence1, sentence2]
    #nltk.word_tokenize(sentenceence)表示对sentenceence进行分词，空格隔开句子中的词
    #对sentences中的每个变量用sentence来表示，对于sentence的分词结果中的每个变量用word来表示
    #temp_texts为二维数组，每一行依次存储每一个句子的分词结果
    temp_texts = [[word for word in word_tokenize(sentence)] for sentence in sentences]
```

以句子1为”A likes to play games. B likes too.“，句子2为”B also likes to play games too.“为例子，分词的结果temp_texts为：

```
[['A', 'likes', 'to', 'play', 'games', '.', 'B', 'likes', 'too', '.'], ['B', 'also', 'likes', 'to', 'play', 'games', 'too', '.']]
```

- 第二步为构造语料库，核心代码以及详细注释如下：

```
    #构建语料库corpus，即句子中所有出现过的单词及标点（不重复）
    #tempList列表用来存储句子中所有的单词及标点（重复）
    tempList = []
    for text in temp_texts:
        tempList += text
    #set() 函数创建一个无序不重复元素集
    corpus = set(tempList)
```

承接上面的例子，tempList的内容为：

```
['A', 'likes', 'to', 'play', 'games', '.', 'B', 'likes', 'too', '.', 'B',
'also', 'likes', 'to', 'play', 'games', 'too', '.']
```

语料库corpus的内容为：

```
{'also', 'play', 'to', '.', 'B', 'A', 'likes', 'too', 'games'}
```

- 第三步为对语料库中的单词及编号建立数字映射，便于构造所需要的向量：

```
    #zip(corpus, range(len(corpus)))将corpus和range(len(corpus))两个列表中的元素
对应起来形成一个个元组
    #dict根据该元组建立一个方便映射的可变容器模型，赋给corpus_dict
    #这么做的目的是将语料库中的单词及标点建立数字映射
    corpus_dict = dict(zip(corpus, range(len(corpus))))
```

承接上面的例子，corpus_dict的内容为：

```
{'also': 0, 'play': 1, 'to': 2, '.': 3, 'B': 4, 'A': 5, 'likes': 6, 'too':
7, 'games': 8}
```

- 第四步是依据bag of word构造句子的向量，核心代码以及详细注释如下：

```
    #定义vectortor_rep函数，得到句子的向量（单词在语料词中的编号，该单词出现的次数）
    def get_vector(text, corpus_dict):
        vector = []
        # 对于语料库中的每一个词
        for key in corpus_dict.keys():
            if key in text:
                vector.append((corpus_dict[key], text.count(key))) # 若句子中
存在该词
            else:
                vector.append((corpus_dict[key], 0)) # 若句子中存在该词

        vector = sorted(vector, key= lambda x: x[0])

        return vector

    #vector1存放句子1的向量
    #vector2存放句子2的向量
    vector1 = get_vector(temp_texts[0], corpus_dict)
    vector2 = get_vector(temp_texts[1], corpus_dict)
```

承接上面的例子，vector1的内容为：

```
[(0, 0), (1, 1), (2, 1), (3, 2), (4, 1), (5, 1), (6, 2), (7, 1), (8, 1)]
```

vector2的内容为：

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 0), (6, 1), (7, 1), (8, 1)]
```

- 第五步是里利用上面提到的方法，使用点积和幅值来计算两个构造向量的余弦值，即得到两个句子的余弦相似度，核心代码以及详细注释如下：

```
    #计算向量的余弦相似度
    #引进math中的sqrt函数
    from math import sqrt
    #函数get_cosine_similarity返回vector1和vector2的向量夹角的余弦值
    def get_cosine_similarity(vector1, vector2):
        inner_product = 0
        square_length_vector1 = 0
        square_length_vector2 = 0
        for tup1, tup2 in zip(vector1, vector2):
            inner_product += tup1[1]*tup2[1]
            square_length_vector1 += tup1[1]**2
            square_length_vector2 += tup2[1]**2

        return
(inner_product/sqrt(square_length_vector1*square_length_vector2))

    #用向量的余弦相似度作为句子的相似度
    usimilarity = get_cosine_similarity(vector1, vector2)
```

  算出来的向量余弦值为0.8504，句子的相似度usimilarity为0.8504

- 值得一提的是，当存在字符串为空字符串时，得到的向量为空向量，计算余弦值时候会出现除以0的程序错误，因此在计算相似度前我们应该先确定空字符串是否存在，如果存在就直接返回（当两个字符串都为空字符串的，则相似度为1，若一个字符串为空，另一个不会空，则相似度为0），，核心代码以及详细注释如下：

```
    ''' 为了避免计算余弦相似度时出现除以
    0的情况，我们要先判断字符串是否为空 '''
    # 两个字符串都为空，相似度为1，一个字符串为空，另一个不为空，相似度为0
    if(len(sentence1)==0 and len(sentence2)==0 ):
        s='\''+sentence1+'\''+' 和 '+'\''+sentence2+'\''+' 的相似度为： %.4f。
\n'%1
        txt.insert(END, s)    # 在文本框中显示运算结果
        return
    elif( len(sentence1)==0 or len(sentence2)==0 ):
        s='\''+sentence1+'\''+' 和 '+'\''+sentence2+'\''+' 的相似度为： %.4f。
\n'%0
        txt.insert(END, s)    # 在文本框中显示运算结果
        input1.delete(0, END)  # 清空非空文本框的输入
        input2.delete(0, END)  # 清空非空文本框的输入
        return
```

3.2 图形化界面的实现：

- 创建窗体，核心代码以及详细注释如下：

```
'''窗体'''
root = Tk() # 初始化一个根窗体实例 root
root.geometry('460x600') # 设置窗体的大小为宽460像素，高600像素
root.title('计算句子对的相似度') #设置窗体的标题文字为“计算句子对的相似度”
```

- 创建两个空白输入框并且设置其在合适的界面位置，用来输入两串用来计算相似度的句子，核心代码以及详细注释如下：

```
'''输入框'''
s1 = StringVar()
s2 = StringVar() #告诉编译器存放的为字符串类型
input1 = Entry(root,textvariable=s1) # 接收字符串输入
input1.place(relx=0.05, rely=0.15, relwidth=0.4, relheight=0.05) # 设置输入框显示的相对位置，以及相对长宽属性
input2 = Entry(root,textvariable=s2)
input2.place(relx=0.55, rely=0.15, relwidth=0.4, relheight=0.05)
```

句子计算完相似度并显示输出后，可以使用delete删除输入框的内容，具体代码如下：

```
input1.delete(0, END)  # 清空输入
input2.delete(0, END)  # 清空输入
```

- 创建标签并且设置其在合适的界面位置，提示用户输入指定信息，核心代码以及详细注释如下：

```
'''标签'''
uLabel = Label(root, text='输入需要计算相似度的句子对') #设置标签
uLabel.place(relx=0.1, rely=0.05, relwidth=0.8, relheight=0.1) # 设置标签显示的相对位置，以及相对长宽属性
```

- 创建按钮并设置其在合适的界面位置，使得用户可以通过鼠标点击按钮的方式触发句子相似度的计算和输出（calculate_sim函数），核心代码以及详细注释如下：

```
'''调用 Calculate_sim()'''
uButton1 = Button(root, text='计算相似度', command=calculate_sim) # 鼠标点击触发函数Calculate the similarity of sentence pairs的执行，其中按钮上的文本为'计算相似度'
uButton1.place(relx=0.1, rely=0.25, relwidth=0.3, relheight=0.075) # 设置按钮显示的相对位置，以及相对长宽属性
```

- 创建按钮并设置其在合适的界面位置，使得用户可以通过鼠标点击按钮的方式触发程序的退出，核心代码以及详细注释如下：

```
'''退出程序'''
uButton2 = Button(root, text='退出', command=root.quit) # 鼠标点击触发程序的退出，其中按钮上的文本为'退出'
uButton2.place(relx=0.6, rely=0.25, relwidth=0.3, relheight=0.075) # 设置按钮显示的相对位置，以及相对长宽属性
```

- 创建文本框用来显示计算的相似度的结果的输出，并设置其在合适的界面位置，核心代码以及详细注释如下：

```
'''文本框'''
txt = Text(root) # 定义文本框
txt.place(relx=0.05,rely=0.4, relwidth=0.9,relheight=0.55) #  设置按钮显示的相对位置，以及相对长宽属性
```

将计算的相似度的结果的输出显示在文本框中，核心代码以及详细注释如下：

```
s='\''+sentence1+'\''+' 和 '+'\''+sentence2+'\''+' 的相似度为： %.4f。\n'%usimilarity
txt.insert(END, s)    # 在文本框中显示运算结果
```

- 实现的界面如下：

4. 实验结果分析

1. 样例一：

   句子对：

```
1. He promises to lend me some books.
2. I promise to lend he some books.
```

   点击计算相似度执行后，界面输出相似度结果，并清空输入框：

2. 样例一：

   句子对：

```
1. 123 321 123
2. 321 123 111
```

   点击计算相似度执行后，界面输出相似度结果，并清空输入框：

3. 样例三：

   句子对：

```
1.
2.
```

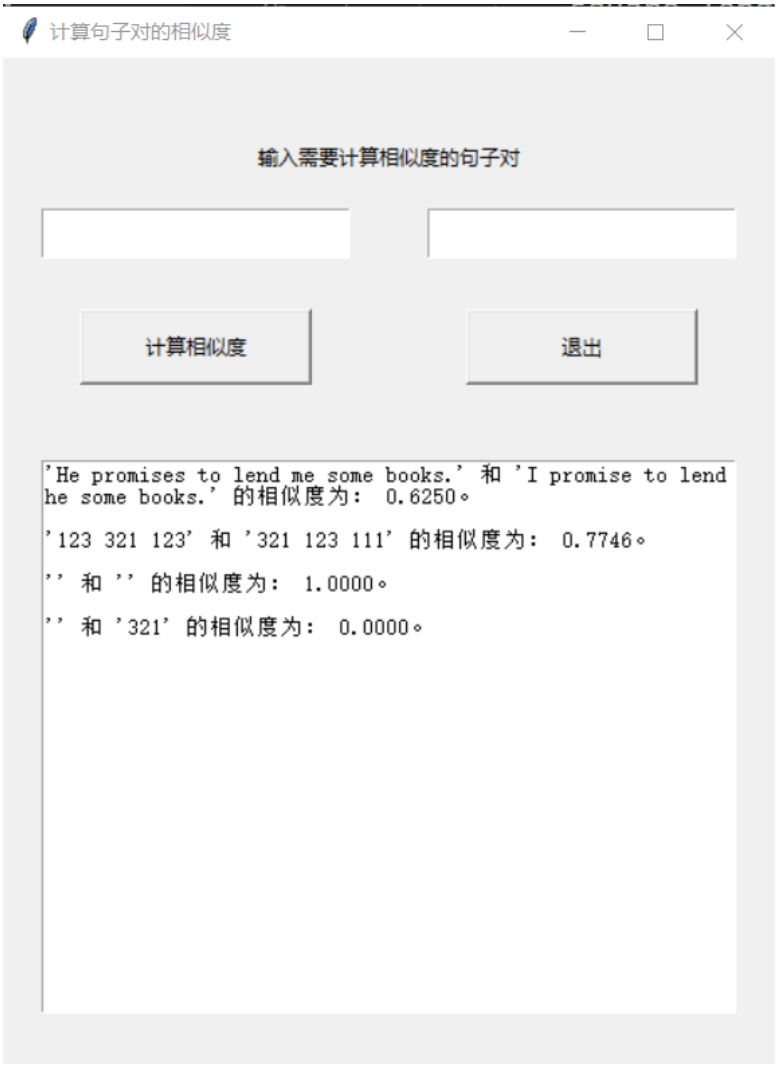
   点击计算相似度执行后，界面输出相似度结果，并清空输入框：

4. 样例四：

   句子对：

```
1.
2. 321
```

   点击计算相似度执行后，界面输出相似度结果，并清空输入框：



5. 同理依次输入以下样例

样例五：

```
1. hello world
2.
```

样例六：

```
1. 输出 相似度 结果，并清空 输入框
2. 输出 相似 的结果，并 清空 输入框
```

样例七：

```
1. He is always ready to help others.
2. He feels a little tired.
```

样例八：

```
1. hello
2. hello world
```

样例九：

```
1. The question is what you want to do.
2. Is Helen in?
```

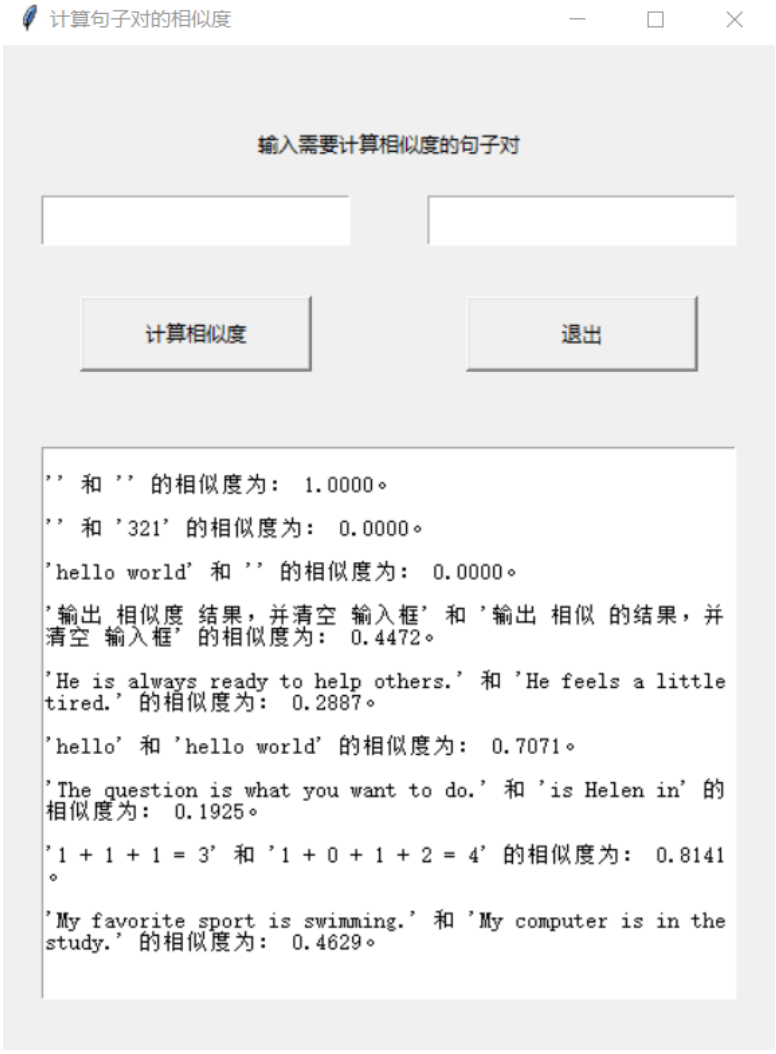
样例十：

```
1. 1 + 1 + 1 = 3
2. 1 + 0 + 1 + 2 = 4
```

样例十一：

```
1. My favorite sport is swimming.
2. My computer is in the study.
```

在执行完以上样例后，界面所有的输出相似度结果，并清空输入框：



点击退出按钮退出程序