《人工智能基础 A》实验报告七

自然语言处理之语义相似度分析

一、实验目的

通过本次实验，了解并实现文本语义相似度计算的常见方法，具体目标要求如下：

1） 理解文本语义相似度计算的基本原理。

2） 实现常见的语义相似度计算方法（如 Word2Vec）等。

3） 对比不同算法在文本相似度计算中的表现。

二、实验内容及要求

实验步骤（仔细阅读，按照步骤完成实验）

1. 完成下列题目（形式任意）

根据下面的表格，计算苹果和大米的余弦相似度和广义 Jaccard 相似度。

	水果	公司	手机	粮食
$R_{橘子}$	0.92	0.23	0.02	0.19
$R_{香蕉}$	0.95	0.13	0.09	0.25
$R_{苹果}$	0.96	0.77	0.85	0.15
$R_{小米}$	0.08	0.81	0.98	0.87
$R_{华为}$	0.02	0.93	0.95	0.01
$R_{大米}$	0.18	0.22	0.05	0.93

2. 词袋模型实践（注意截图）

1. 构建两句话的词袋模型并计算相似度

a) 准备工作

首先导入所需的模块，然后准备需要计算相似度的句子。在这个例子中，我们将使用 CountVectorizer 来构建词袋模型，并通过 cosine_similarity 来计算句子之间的相似度。大家可以自行输入两个相似但不相同的两个中文句子。

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity


# 输入的两个句子
sentence1 = "……"
sentence2 = "……"
```

b) 分词

下面对中文句子进行分词。

```
import jieba
# 使用 jieba 进行分词
def jieba_cut(sentence):
    return " ".join(jieba.cut(sentence))


sentence1_cut = jieba_cut(sentence1)
sentence2_cut = jieba_cut(sentence2)
```

c) 构建词袋模型

首先需要把句子转化为词袋模型，每个句子的词袋模型是一个向量，向量的每个维度对应词汇表中的一个词，值是该词在句子中的出现频率。

```
# 使用 CountVectorizer 构建词袋模型
vectorizer = CountVectorizer()
# 将两个句子转换为词袋向量
X = vectorizer.fit_transform([sentence1_cut, sentence2_cut])
```

d) 计算余弦相似度

通过计算两个向量之间的余弦相似度，我们可以衡量两个句子在词袋空间中的相似度。

```
# 计算余弦相似度
cosine_sim = cosine_similarity(X[0:1], X[1:2])


# 输出相似度
print(f"两个句子的余弦相似度为: {cosine_sim[0][0]:.4f}")
```

2. 参考上述代码，尝试构建两句话以上的词袋模型并计算相似度

3. 使用 Word2Vec 计算文本相似度（注意截图）

1. 准备工作

首先导入所需库。我们利用 jieba 库来对中文句子进行分词，它能够将句子分解成一个个词语；gensim.models.Word2Vec 用来训练 Word2Vec 模型；cosine_similarity 用于计算句子之间的余弦相似度。

```
import jieba
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
```

接下来准备句子数据，可以用下面提供的，也可以自行定义。

```
sentences = [
    "我喜欢在清晨喝一杯热茶，享受安静的时光。",
    "今天是我生日，我准备和家人一起庆祝。",
    "科技改变了我们的生活，尤其是人工智能的发展。",
    "早晨的阳光洒在大地上，给一切带来了温暖。",
    "我喜欢在每天下午来一杯咖啡，它能让我更加集中精神。",
    "每当下雨时，我喜欢在窗前静静地看着雨滴落下。",
    "昨晚我和朋友一起去看了一场电影，感觉很放松。",
    "电脑和手机已经成为现代社会不可或缺的一部分。",
    "他每天都在健身房锻炼，已经养成了好习惯。",
    "今天是我的生日，但我与朋友约好了一起在海底捞庆祝。",
    "我喜欢读科幻小说，因为它们带我进入了另一个未知的世界。"
]
```

2. 分词

- 将输入的句子列表进行中文分词，并输出每个句子的分词结果。

```
# 使用 jieba 对句子进行分词
def cut_sentences(sentences):
    return [list(jieba.cut(sentence)) for sentence in sentences]

tokenized_sentences = cut_sentences(sentences)

# 打印分词后的结果
for sentence in tokenized_sentences:
    print(" ".join(sentence))
```

3. 模型训练

Word2Vec 是一个将单词转换为向量的模型，训练时会从文本中学习词与词之间的语义关系。

```
# 训练 Word2Vec 模型
model = Word2Vec(tokenized_sentences, size=100, window=5, min_count=1, sg=0)
```

4. 计算句子向量

计算每个句子的词向量平均值，作为句子的向量表示。

```
import numpy as np


# 计算句子向量
def sentence_vector(tokens):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    return np.mean(vectors, axis=0) if vectors else np.zeros(100)


# 获取每个句子的词向量
sentence_vectors = [sentence_vector(sentence) for sentence in
tokenized_sentences]
```

5. 计算相似度

- 利用 cosine_similarity 来计算句子之间的余弦相似度。

```
# 计算余弦相似度
from sklearn.metrics.pairwise import cosine_similarity
cosine_sim = cosine_similarity(sentence_vectors)
```

-

6. 可视化结果

可以用热图来可视化结果，热图中的每个格子表示两个句子之间的余弦相似度。

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
sns.heatmap(cosine_sim, annot=True, cmap="coolwarm", xticklabels=[f"句子{i+1}" for i in range(len(sentences))], yticklabels=[f"句子{i+1}" for i in range(len(sentences))], fmt=".2f")
plt.title("句子余弦相似度热图")
plt.show()
```

作业上交内容与事项：

1. 按照要求完成实验并将关键步骤实验结果进行截图记录，注意文档工整。
2. 请在截止日期内提交，若逾期提交，成绩会被适当打折。

本次作业上交内容：

- 实验报告文档(.docx)

三、实验感受与记录

3-1、实验感受（总结实验过程中的收获或疑问）

本次实验让我对文本语义相似度计算有了更深入的理解和实践体验。通过实现基于表格特征向量的余弦相似度和广义 Jaccard 相似度计算，我直观地感受到了这两种经典相似度度量方式的计算原理和适用场景，理解了如何将结构化数据转化为向量并进行相似性比较。

词袋模型实践让我了解了这种简单高效的文本表示方法，虽然它忽略了词序和语义信息，但在某些场景下仍不失为一种快速有效的解决方案。Word2Vec 模型的应用，从中文分词、模型训练、句子向量的生成，到最终的相似度计算

和热图可视化，我完整地体验了如何利用词嵌入技术来捕捉文本之间更深层次的语义关联。对比词袋模型，Word2Vec 能够更好地理解词语的含义，从而在语义层面计算句子相似度，这在结果热图中得到了清晰的展示。

实验也让我意识到，高质量的语料库和恰当的模型参数选择对于提升 Word2Vec 等模型的表现至关重要。总的来说，这次实验不仅巩固了理论知识，更锻炼了动手能力，让我对自然语言处理中语义相似度分析技术有了更全面的认识。

3-2、实验记录（截取一些操作界面放置于如下）

1.计算苹果和大米的余弦相似度和广义 Jaccard 相似度。

余弦相似度：衡量两个向量方向上的一致性，计算公式为：

$$CosineSimilarity(A,B)=\frac{A\cdot B}{\|A\|\|B\|}=\frac{\sum_{i=1}^{n}A_iB_i}{\sqrt{\sum_{i=1}^{n}A_i^2}\sqrt{\sum_{i=1}^{n}B_i^2}}$$

广义 Jaccard 相似度：衡量两个集合的相似程度，对于向量，其计算公式为：

$$JaccardSimilarity(A,B)=\frac{A\cdot B}{\|A\|^2+\|B\|^2-A\cdot B}=\frac{\sum_{i=1}^{n}A_iB_i}{\sum_{i=1}^{n}A_i^2+\sum_{i=1}^{n}B_i^2-\sum_{i=1}^{n}A_iB_i}$$

我使用 python 代码实现计算，结果如下：

```
苹果和大米的特征向量:
R_苹果 = [0.96 0.77 0.85 0.15]
R_大米 = [0.18 0.22 0.05 0.93]
苹果和大米的余弦相似度: 0.3581
苹果和大米的广义Jaccard相似度: 0.1953
```

2. 词袋模型实践

- 准备工作：导入 CountVectorizer 用于构建词袋模型和 cosine_similarity 用于计算余弦相似度 。导入 jieba 用于中文分词。
- 定义句子：创建了包含 4 个中文句子的列表。
- 分词：定义一个函数，使用 jieba.cut 对每个句子进行分词，并将分词结果用空格连接。
- 构建词袋模型：实例化 CountVectorizer，然后使用 fit_transform 方法将分词后的句子列表转换为词频矩阵（词袋向量）。

- 计算余弦相似度：使用 cosine_similarity 函数计算句子向量两两之间的余弦相似度，得到一个相似度矩阵；然后将相似度矩阵按照更易读的方式输出。

代码运行结果如下：

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\DELL\AppData\Local\Temp\jieba.cache
Loading model cost 0.548 seconds.
Prefix dict has been built successfully.
分词结果:
我 喜欢 吃 苹果 ， 也 喜欢 香蕉 。
今天天气 真 好 ， 适合 出去玩 。
小明 喜欢 在 公园 里 跑步 ， 小红 喜欢 在 家里 看书 。
苹果 是 一种 常见 的 水果 ， 营养 丰富 。
------------------------------
句子间余弦相似度矩阵:
[[1.         0.         0.51639778 0.16666667]
 [0.         1.         0.         0.        ]
 [0.51639778 0.         1.         0.        ]
 [0.16666667 0.         0.         1.        ]]
句子'我喜欢吃苹果，也喜欢香蕉。' 与 句子'今天天气真好，适合出去玩。' 的余弦相似度为: 0.0000
句子'我喜欢吃苹果，也喜欢香蕉。' 与 句子'小明喜欢在公园里跑步，小红喜欢在家里看书。' 的余弦相似度为: 0.5164
句子'我喜欢吃苹果，也喜欢香蕉。' 与 句子'苹果是一种常见的水果，营养丰富。' 的余弦相似度为: 0.1667
句子'今天天气真好，适合出去玩。' 与 句子'小明喜欢在公园里跑步，小红喜欢在家里看书。' 的余弦相似度为: 0.0000
句子'今天天气真好，适合出去玩。' 与 句子'苹果是一种常见的水果，营养丰富。' 的余弦相似度为: 0.0000
句子'小明喜欢在公园里跑步，小红喜欢在家里看书。' 与 句子'苹果是一种常见的水果，营养丰富。' 的余弦相似度为: 0.0000
```

3. 使用 Word2Vec 计算文本相似度

- 准备工作：导入 jieba 用于分词，Word2Vec 来自 gensim.models 用于训练词向量模型，cosine_similarity 来自 sklearn.metrics.pairwise 用于计算相似度，numpy 用于数值计算，seaborn 和 matplotlib.pyplot 用于绘图。
- 定义句子：提供一个包含多个中文句子的列表，同实验指导中的示例句子。
- 分词：定义一个函数，使用 jieba.cut 对每个句子进行分词，结果是词语列表的列表。

  分词结果：

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\DELL\AppData\Local\Temp\jieba.cache
分词后的结果:
句子1: 我 喜欢 在 清晨 喝一杯 热茶 ， 享受 安静 的 时光 。
句子2: 今天 是 我 生日 ， 我 准备 和 家人 一起 庆祝 。
句子3: 科技 改变 了 我们 的 生活 ， 尤其 是 人工智能 的 发展 。
句子4: 早晨 的 阳光 洒 在 大 地上 ， 给 一切 带来 了 温暖 。
句子5: 我 喜欢 在 每天 下午 来 一杯 咖啡 ， 它 能 让 我 更加 集中 精神 。
句子6: 每当 下雨 时 ， 我 喜欢 在 窗前 静静地 看着 雨滴 落下 。
句子7: 昨晚 我 和 朋友 一起 去 看 了 一场 电影 ， 感觉 很 放松 。
句子8: 电脑 和 手机 已经 成为 现代 社会 不可或缺 的 一部分 。
句子9: 他 每天 都 在 健身房 锻炼 ， 已经 养成 了 好 习惯 。
句子10: 今天 是 我 的 生日 ， 但 我 与 朋友 约 好 了 一起 在 海底 捞 庆祝 。
句子11: 我 喜欢 读 科幻小说 ， 因为 它们 带 我 进入 了 另 一个 未知 的 世界 。
------------------------------
```

- 模型训练：使用分词后的句子列表训练 Word2Vec 模型。参数包括 size (向量维度)，window (上下文窗口大小)，min_count (忽略词频低于此值的词)，sg (0 表示 CBOW 模型，1 表示 Skip-gram 模型)。
- 计算句子向量：定义一个函数，通过获取句子中所有词语的 Word2Vec 向量并取其平均值，来计算整个句子的向量表示。如果句子中的词都不在模型的词汇表中，则返回一个零向量。
- 计算相似度：使用 cosine_similarity 计算所有句子向量之间的余弦相似度，生成相似度矩阵。

  打印相似度矩阵：

```
Loading model cost 0.516 seconds.
Prefix dict has been built successfully.
句子余弦相似度矩阵:
[[1.    0.214 0.366 0.33  0.376 0.352 0.232 0.262 0.248 0.339 0.457]
 [0.214 1.    0.138 0.142 0.421 0.232 0.472 0.185 0.196 0.669 0.377]
 [0.366 0.138 1.    0.396 0.118 0.203 0.086 0.187 0.076 0.27  0.323]
 [0.33  0.142 0.396 1.    0.305 0.347 0.341 0.144 0.266 0.382 0.355]
 [0.376 0.421 0.118 0.305 1.    0.334 0.391 0.052 0.362 0.468 0.479]
 [0.352 0.232 0.203 0.347 0.334 1.    0.362 0.051 0.344 0.343 0.381]
 [0.232 0.472 0.086 0.341 0.391 0.362 1.    0.17  0.326 0.537 0.447]
 [0.262 0.185 0.187 0.144 0.052 0.051 0.17  1.    0.201 0.301 0.252]
 [0.248 0.196 0.076 0.266 0.362 0.344 0.326 0.201 1.    0.389 0.327]
 [0.339 0.669 0.27  0.382 0.468 0.343 0.537 0.301 0.389 1.    0.577]
 [0.457 0.377 0.323 0.355 0.479 0.381 0.447 0.252 0.327 0.577 1.   ]]
------------------------------
```

- 可视化结果：使用 seaborn 的 heatmap 函数将相似度矩阵可视化为热图，更直观地展示句子间的相似程度。
  可视化结果图：