Report of Deep Learning for Natural Langauge Processing

Word Embedding

Jiayu Cui
cuijiayu_2001@163.com

Abstract

利用给定语料库（金庸小说语料），利用 1～2 种神经语言模型（如：基于 Word2Vec，LSTM，GloVe 等模型）来训练词向量，通过计算词向量之间的语意距离、某一类词语的聚类、某些段落直接的语意关联、或者其他方法来验证词向量的有效性。

Methodology

Word2vec 是 2013 年被 Mikolov 提出来的词向量训练算法，Word2vec 有两种具体实现方式：连续词袋模型 CBOW 和 Skip-gram，如图 1 所示。

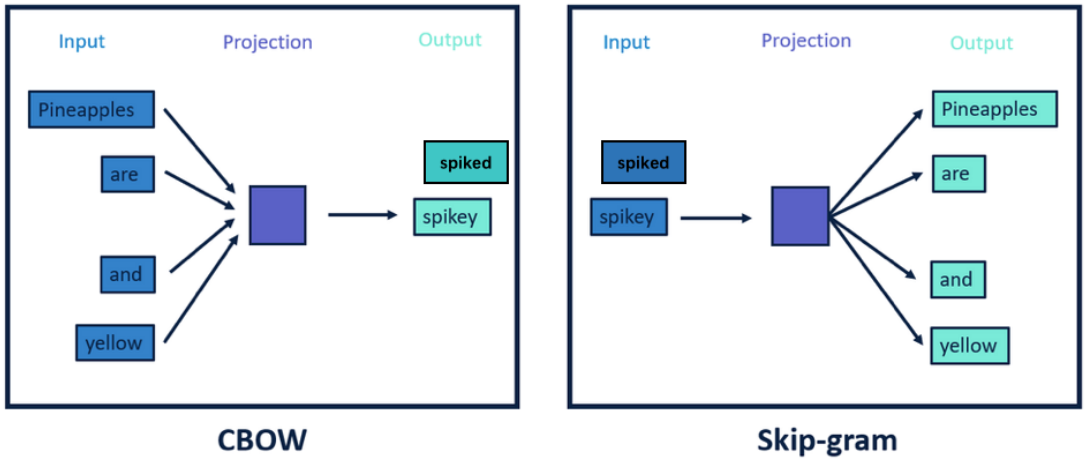

图 1 CBOW 和 Skip-gram 的对比

图 1 中使用了这句话作为例子：Pineapples are spiked and yellow，在这句话中假设中心词是 spiked，这个单词的上下文是其他单词:Pineapples are and yellow。

连续词袋模型 CBOW 的建模方式是使用上下文单词来预测 spiked 这个单词，当然图片上展示的是 spikey，相当于是预测错了。Skip-gram 正好反过来，它是通过中心词来预测上下文。

一般来说，CBOW 比 Skip-garm 训练快且更加稳定一些，然而，Skip-garm 不会刻意地回避生僻词（即出现频率比较低的词），比 CBOW 能够更好地处理

生僻词。

在实验中，我们将以 Skip-garm 的方式进行词向量的训练过程。（即在配置参数时选择 sg=1，表示使用 Skip-gram 模型，该模型对小型数据集较为有效。）Skip-gram 网络结构如图 2 所示。

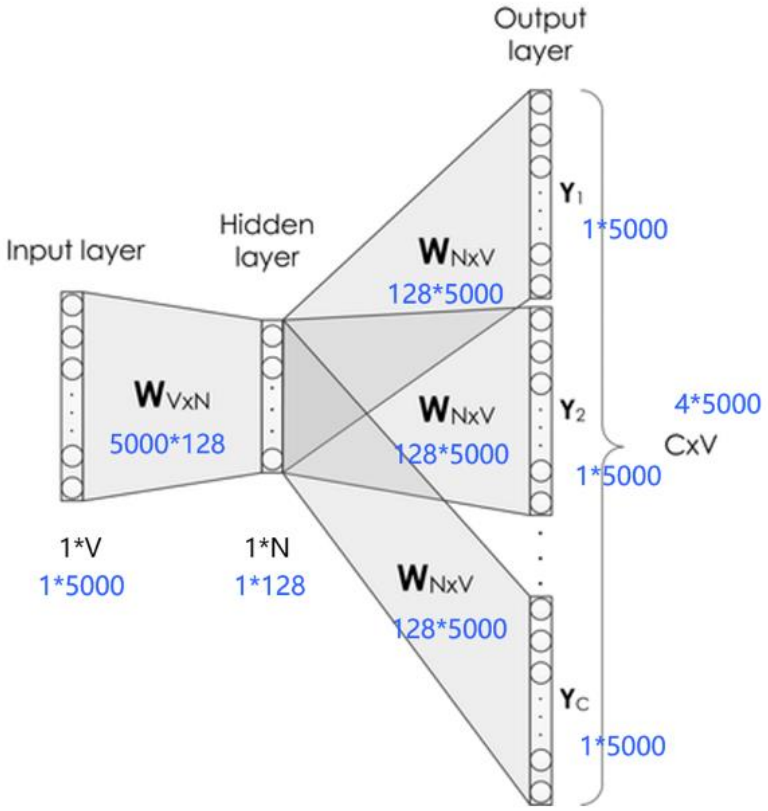

图 2 Skip-gram 网络结构

Skip-gram 的网络结构共包含三层：输入层，隐藏层和输出层。它的处理步骤是这样的：

1) 输入层接收 shape 为[1,V]的 one-hot 向量 x，其中 V 代表词表中单词的数量，这个 one-hot 向量就是上边提到的中心词。
2) 隐藏层包含一个 shape 为$[V,N]$的参数矩阵$W1$，其中这个N代表词向量的维度，$W1$ 就是 word embedding 矩阵，即我们要学习的词向量。将输入的 one-hot 向量x与$W1$ 相乘，便可得到一个 shape 为$[1,N]$的向量，即该输入单词对应的词向量e。
3) 输出层包含一个 shape 为$[N,V]$的参数矩阵$W2$，将隐藏层输出的e与$W2$ 相乘，便可以得到 shape 为$[1,V]$的向量r，内部的数值分别代表每个候选词的打分，使用 softmax 函数，对这些打分进行归一化，即得到中心词的预测各个单词的概率。

Experimental Studies

S1: Word2vec 训练

1) 数据预处理：语料库由从文本数据集中提取和预处理的句子组成，去除了噪声和无关的词元。文本数据经过分词，并去除了停用词。

2) 使用 Gensim 库训练 Word2Vec 模型，采用 Skip-gram 方法，并设定特定参数，如图 3 所示。

```
def train_word2vec_model(sentences, sg=0, vector_size=200, window=5, min_count=5, epochs=50, negative=10):
    """
    训练Word2Vec模型

    参数:
        sentences (list): 句子列表
        sg (int): Skip-gram（1）或CBOW（0）模型
        vector_size (int): 词向量维度
        window (int): 窗口大小
        min_count (int): 忽略频率低于此值的单词
        epochs (int): 训练轮数
        negative (int): 负采样数量

    返回:
        gensim.models.Word2Vec: 训练好的Word2Vec模型
    """
    model = Word2Vec(sentences, vector_size=vector_size, window=window, min_count=min_count, epochs=epochs, negative=negative, sg=sg)
    return model


# 模型
model = train_word2vec_model(text,sg=1, vector_size=200, window=5, min_count=5, epochs=50,negative=10)
```

图 3 Word2Vec 模型参数

3) 训练过程包括以下步骤：初始化：使用指定参数初始化 Word2Vec 模型。构建词汇表：模型扫描语料库，构建满足 min_count 阈值的所有唯一单词的词汇表。训练：模型在语料库上迭代指定的轮数，使用随机梯度下降和反向传播更新词向量的权重。负采样：在训练期间，采用负采样以提高效率，模型根据正（观察到的）和负（随机采样的）词对更新权重。

参数选择显著影响模型的性能。选择 Skip-gram 模型是因为其在小型数据集上的表现良好，同时 200 维的词向量在捕捉详细关系和计算效率之间取得了平衡。窗口大小为 5，确保考虑到局部和稍微广泛的上下文信息。负采样设置为 10，有助于在保持模型准确性的同时高效地更新权重

S2：词义相似度

使用预先训练的 Word2Vec 模型来获取每个人物的词向量，然后找到与每个人物最相似的词语。代码如图 4 所示。

```
# 查找相似词语并生成词云
characters = ['令狐冲', '张无忌', '杨过', '郭靖', '段誉']
fig, axs = plt.subplots(2, 3, figsize=(15, 10), dpi=300)
zhfont = matplotlib.font_manager.FontProperties(fname=FONT_PATH, size=12)

for ax, character in zip(axs.flatten(), characters):
    print(f"与 {character} 最相似的词语:")
    similar_words = find_similar_words(model, character, topn=10)
    for word, similarity in similar_words:
        print(f"{word}: {similarity}")
    # 生成词云
    words_freq = {word: similarity for word, similarity in similar_words}
    generate_wordcloud(words_freq, ax, character, zhfont)

# 删除多余的子图
for i in range(len(characters), len(axs.flatten())):
    fig.delaxes(axs.flatten()[i])

plt.tight_layout()
plt.show()
✓ 24.3s
```

图 4 词义相似度代码

为每个人物及其相似词语生成了词云和条形图。词云显示了每个词语的相对频率，如图 5 所示。

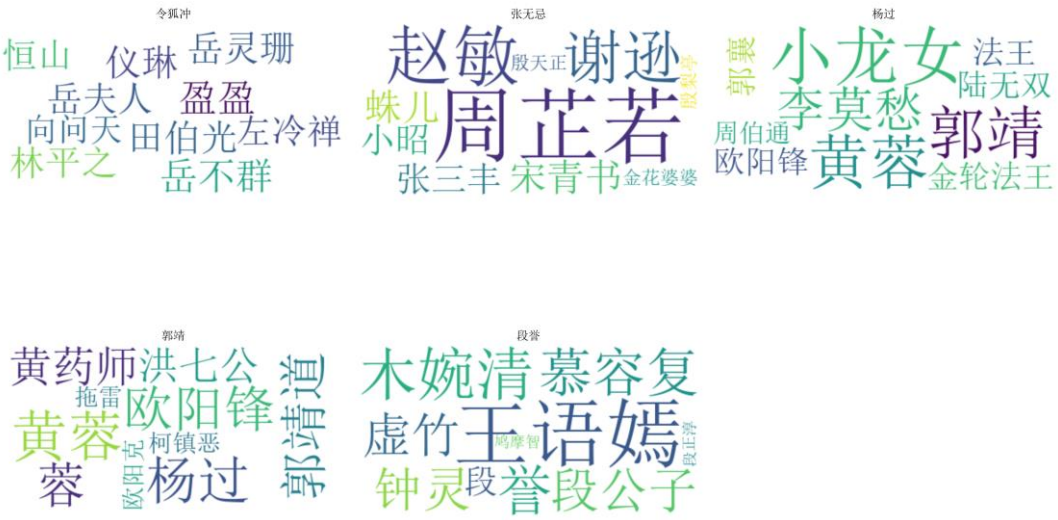

图 5 词云图

条形图显示了每个词语的具体相似度值。如图 6 所示。

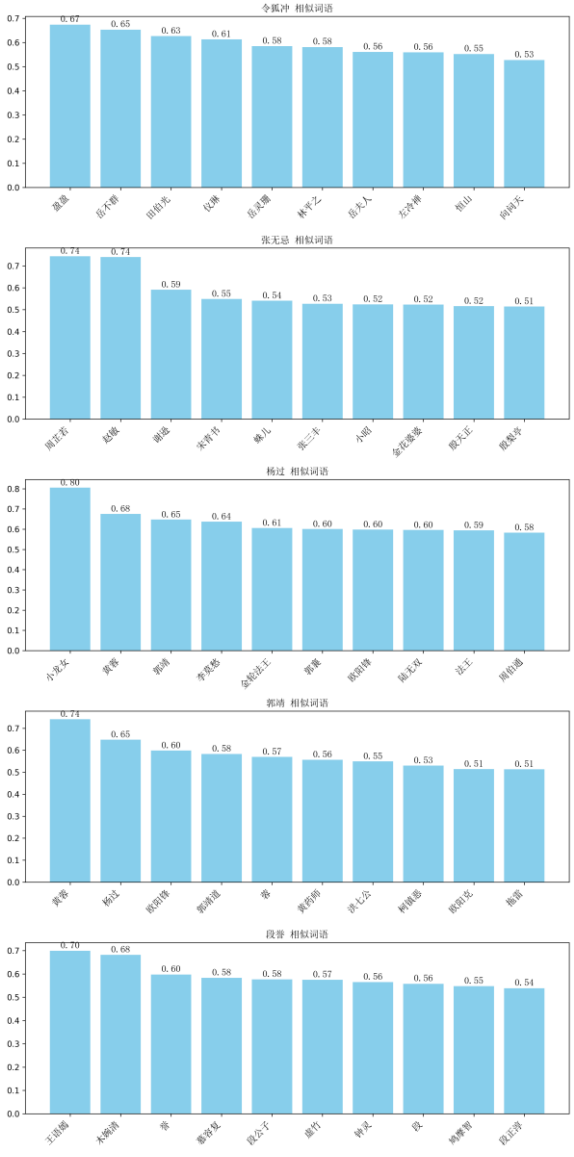

图 6 条形图

分析生成的词云和条形图，从这些结果可以看到一些有趣的关联性，这些关联反映了金庸小说人物之间的关系和故事情节的相似性。以下是对这些结果的分析：

与令狐冲相关性高的词：

1) 盈盈 (0.674): 任盈盈，令狐冲的情人，两人关系密切，故事中有很多互动。
2) 岳不群 (0.653): 令狐冲的师父，尽管后来的关系变得紧张，但在早期的情节中有很多交集。
3) 田伯光 (0.626): 田伯光和令狐冲在故事中的互动也相对较多。
4) 仪琳 (0.613): 仪琳喜欢令狐冲，两人关系比较复杂。
5) 岳灵珊 (0.585): 令狐冲的师妹，也是他的初恋。
6) 林平之 (0.581): 林平之和令狐冲有较多的故事情节交集，尤其在后来成为敌人。
7) 岳夫人 (0.561): 岳不群的妻子，令狐冲的师母。
8) 左冷禅 (0.559): 左冷禅和令狐冲在故事中有对抗。
9) 恒山 (0.552): 令狐冲后期成为恒山派的掌门。
10) 向问天 (0.528): 向问天和令狐冲有合作关系，共同对抗敌人。

与杨过相关性高的词：

杨过

1) 小龙女 (0.805): 小龙女是杨过的师父和爱人，两人关系非常密切。
2) 黄蓉 (0.676): 黄蓉是杨过的长辈，有很多故事情节的交集。
3) 郭靖 (0.647): 郭靖是杨过的长辈，两人有很多互动，尤其是在郭襄的情节中。
4) 李莫愁 (0.637): 李莫愁是小龙女的师姐，与杨过有较多交集。
5) 金轮法王 (0.606): 金轮法王是杨过的敌人，两人有较多对抗。
6) 郭襄 (0.601): 郭襄是郭靖和黄蓉的女儿，对杨过有爱慕之情。
7) 欧阳锋 (0.598): 欧阳锋是杨过的义父，两人关系特殊。
8) 陆无双 (0.596): 陆无双对杨过有爱慕之情，两人关系较为复杂。
9) 法王 (0.594): 这里是**由于分词错误导致的**，法王应该是指金轮法王。
10) 周伯通 (0.583): 周伯通是杨过的长辈，两人有一些互动。

这些相似词语的结果符合金庸小说中的人物关系和故事情节：

1) 亲密关系：相似词语中大多是与主角有亲密关系的人物，如令狐冲和任盈盈，张无忌和赵敏、小昭，杨过和小龙女，郭靖和黄蓉，段誉和王语嫣等。
2) 对手或敌人：也包括了一些主要对手或敌人，如郭靖和欧阳锋，杨过和金轮法王，段誉和鸠摩智等。
3) 家人或师父：还有一些家人或师父，如张无忌和谢逊、张三丰，郭靖和洪七公、黄药师，段誉和段正淳等。

总的来说，这些结果显示了词向量模型在捕捉人物关系和情节方面的有效性。所建立的 Word2Vec 模型能够识别出小说中的重要人物及其复杂的关系网络。

S3：聚类分析

选取金庸小说中的主要人物，定义了每个人物所属的书籍（来自《射雕英雄传》、《神雕侠侣》、《倚天屠龙记》、《天龙八部》、《笑傲江湖》五部）。

采用预先训练的 Word2Vec 模型来获取每个人物的词向量，然后使用 K-means 算法对这些词向量进行聚类，可视化聚类结果。代码如图 7 所示。

```
# 定义每个人物所属的书籍
character_books = {
    '郭靖': '射雕英雄传', '黄蓉': '射雕英雄传', '杨过': '神雕侠侣', '赵敏': '倚天屠龙记',
    '张无忌': '倚天屠龙记', '张翠山': '倚天屠龙记', '郭襄': '神雕侠侣', '小龙女': '神雕侠侣',
    '武三通': '射雕英雄传', '周芷若': '倚天屠龙记', '乔峰': '天龙八部', '段誉': '天龙八部',
    '虚竹': '天龙八部', '萧峰': '天龙八部', '木婉清': '天龙八部', '令狐冲': '笑傲江湖',
    '林平之': '笑傲江湖', '岳不群': '笑傲江湖', '任我行': '笑傲江湖', '东方不败': '笑傲江湖',
    '谢逊': '倚天屠龙记', '宋青书': '倚天屠龙记', '梅超风': '射雕英雄传', '慕容复': '天龙八部',
    '段正淳': '天龙八部', '灭绝师太': '倚天屠龙记', '岳灵珊': '笑傲江湖', '柯镇恶': '射雕英雄传',
    '欧阳锋': '射雕英雄传', '王语嫣': '天龙八部', '鸠摩智': '天龙八部', '洪七公': '射雕英雄传',
    '裘千仞': '射雕英雄传', '一灯大师': '射雕英雄传', '周伯通': '射雕英雄传', '陆无双': '神雕侠侣',
    '金轮法王': '神雕侠侣', '赵志敬': '神雕侠侣', '李莫愁': '神雕侠侣'
}

# 聚类分析
invest_words = [
    '郭靖', '黄蓉', '杨过', '赵敏', '张无忌', '张翠山', '郭襄', '小龙女', '武三通', '周芷若',
    '乔峰', '段誉', '虚竹', '萧峰', '木婉清', '令狐冲', '林平之', '岳不群', '任我行',
    '东方不败', '谢逊', '宋青书', '梅超风', '慕容复', '段正淳', '灭绝师太', '岳灵珊', '柯镇恶',
    '欧阳锋', '王语嫣', '鸠摩智', '洪七公', '裘千仞', '一灯大师', '周伯通', '陆无双', '金轮法王',
    '赵志敬', '李莫愁'
]

word_vectors, kmeans = cluster_words(model, invest_words)
visualize_clusters(invest_words, word_vectors, kmeans)
```

图 7 聚类分析代码

为了评估聚类效果，计算了每个聚类的纯度。纯度是一个衡量聚类质量的指标，它的值越高，说明聚类的效果越好。首先获取每个聚类中的元素，然后统计这些元素所属的书籍的数量。纯度被定义为一个聚类中最多的书籍的数量除以该聚类中的元素总数。每个聚类的纯度如图 8 所示。

```
Cluster 0 purity: 1.00, elements: ['赵敏', '张无忌', '周芷若', '宋青书', '灭绝师太'], dominant book: 倚天屠龙记
Cluster 1 purity: 0.67, elements: ['郭靖', '黄蓉', '杨过', '郭襄', '小龙女', '梅超风', '柯镇恶', '欧阳锋', '洪七公', '裘千仞', '周伯通', '李莫愁'], dominant
Cluster 2 purity: 0.42, elements: ['张翠山', '武三通', '令狐冲', '林平之', '岳不群', '任我行', '谢逊', '岳灵珊', '一灯大师', '陆无双', '金轮法王', '赵志敬'],
Cluster 3 purity: 0.80, elements: ['段誉', '虚竹', '木婉清', '东方不败', '段正淳'], dominant book: 天龙八部
Cluster 4 purity: 1.00, elements: ['乔峰', '萧峰', '慕容复', '王语嫣', '鸠摩智'], dominant book: 天龙八部
```

图 8 聚类纯度结果

Cluster 0 和 Cluster 2 的纯度为 1，显示了模型对某些书中的人物识别非常准确。Cluster 1 和 Cluster 3 的纯度小于 1，尽管有混合，但主导书籍的人物仍占多数，表明模型能在一定程度上区分这些书中的人物，但有时会受到不同书中人物特征相似性的影响。

聚类结果如图 9 所示。

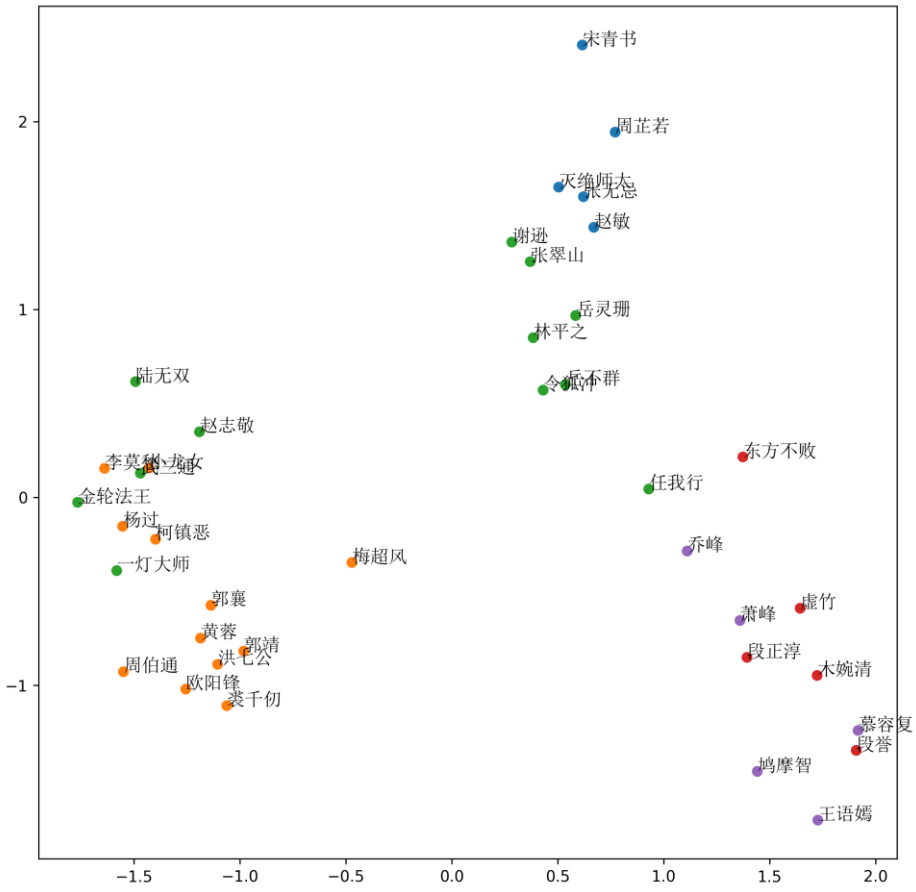

图 9 聚类结果图

S4：段落相似度

从相同文章中抽取段落，得到的段落相似度如图所示。句子 1 和句子 2 的相似度为 0.74，句子 1 和句子 3 的相似度为 0.76，句子 2 和句子 3 的相似度为 0.70。这些句子都来自《倚天屠龙记》。

这表明在同一部小说中，不同句子之间的相似度较高，反映了相似的语言风格和内容主题。代码及结果如图 10 所示。

```
# 从原文中加载句子
sentences_gjx = load_sentences('D:\doc\课业\研一下\DeepNLP\hw3\code\data\倚天屠龙记.txt')
sentences_ygr = load_sentences('D:\doc\课业\研一下\DeepNLP\hw3\code\data\倚天屠龙记.txt')
sentences_lfh = load_sentences('D:\doc\课业\研一下\DeepNLP\hw3\code\data\倚天屠龙记.txt')

# 随机抽取一些示例句子
sentence1 = get_random_sentence(sentences_gjx)
sentence2 = get_random_sentence(sentences_ygr)
sentence3 = get_random_sentence(sentences_lfh)
# 计算句子之间的相似度
similarity1 = compute_sentence_similarity(model, sentence1, sentence2)
similarity2 = compute_sentence_similarity(model, sentence1, sentence3)
similarity3 = compute_sentence_similarity(model, sentence2, sentence3)

print(f"句子1: {sentence1}")
print(f"句子2: {sentence2}")
print(f"句子3: {sentence3}")
print(f"句子1 和 句子2 的相似度: {similarity1:.2f}")
print(f"句子1 和 句子3 的相似度: {similarity2:.2f}")
print(f"句子2 和 句子3 的相似度: {similarity3:.2f}")
```

✓ 0.0s Python

```
句子1: 渡厄道："阳顶天的仇怨已于昨晚化解，罗汉像的事今日也揭过了，好得很，好得很。张教主，你们几位上来动手？"杨逍等见三僧身形矮小瘦削，嵌在松树干中，便像是三具僵尸人干，但几句话
句子2: 朱长龄对张无忌一直容让，只不过不肯死心，盼望最后终能骗动了他，带领自己前往冰火岛去，这时眼见生路已断，而所以陷此绝境，全是为了这小子，一口怨气哪里消得下去？双眼中如要喷出
句子3: 海沙派众私枭瞧那铁倒的同伴时，但见他蜷成一团，早已气绝。各人又惊又怒，有几人放下担子向白袍客去路急追，但那人奔行如飞，黑暗之中哪里还寻得到他的踪影。俞岱岩心道："这白袍客
句子1 和 句子2 的相似度: 0.74
句子1 和 句子3 的相似度: 0.76
句子2 和 句子3 的相似度: 0.70
```

图 10 同一小说段落相似度分析

从不同文章中抽取段落，得到的段落相似度如图所示。假设我们从《倚天屠龙记》、《鹿鼎记》和《笑傲江湖》中各随机抽取一个句子，并计算其相似度。代码及结果如图 11 所示。

图 11 不同小说段落相似度分析

句子 1 和 句子 2 的相似度为 0.61，句子 1 和 句子 3 的相似度为 0.77，句子 2 和句子 3 的相似度为 0.63。这些相似度值比来自同一部小说的句子相似度明显更低，说明不同小说之间的句子相似度较低。

通过这些相似度计算，我们可以验证 Word2Vec 词向量模型在捕捉文本语义上的有效性。

References

[1]https://paddlepedia.readthedocs.io/en/latest/tutorials/sequence_model/word_representation/word2vec.html