Report of Deep Learning for Natural Langauge Processing

Yiwen Cui
wutcyw@163.com

Abstract

本文旨在研究自然语言处理（NLP）中的词向量技术，通过对金庸小说语料库进行处理，采用 Word2Vec 和 FastText 两种词向量模型进行训练，并对其进行验证和比较。实验表明，两种模型在不同任务中的表现各有优劣，Word2Vec 在词对相似度计算和词向量聚类上表现出色，而 FastText 在处理未见词和子词信息方面具有优势。本文的研究结果为进一步改进和应用词向量模型提供了重要参考。

Introduction

自然语言处理（Natural Language Processing，NLP）是计算机科学与人工智能领域的重要分支，旨在实现计算机对人类语言的理解、处理和生成。词向量（Word Embeddings）是 NLP 中的一种核心技术，它将词语映射到高维向量空间中，使得计算机能够更好地理解和处理文本数据。向量技术的背景可以追溯到分布式表示学习的概念，这一概念的基础是著名的分布假设（Distributional Hypothesis）：词语的意义由其上下文决定，即具有相似上下文的词语在语义上也应当是相似的。为了实现这一目标，研究者们提出了多种方法，其中包括经典的词袋模型（Bag of Words, BoW）和 TF-IDF（Term Frequency-Inverse Document Frequency）等。然而，这些方法存在一些局限性，例如无法捕捉词语之间的语义关系、维度高且稀疏等问题。

为了解决上述问题，词向量技术应运而生。其中，Word2Vec 模型是词向量技术的代表之一。Word2Vec 由 Tomas Mikolov 等人在 2013 年提出，它通过神经网络模型将词语嵌入到一个低维的连续向量空间中，使得语义相似的词语在向量空间中距离更近。

除了 Word2Vec 之外，还有许多其他的词向量模型，例如 GloVe（Global Vectors for Word Representation）、FastText 以及近年来流行的 BERT（Bidirectional Encoder Representations from Transformers）等。这些模型在不同的应用场景中表现出了卓越的效果，并在文本分类、情感分析、机器翻译等任务中得到了广泛应用。

本次实验的目的是利用金庸小说语料库，通过 Word2Vec 等神经语言模型训练词向量，并验证其有效性。具体而言，本实验旨在实现以下几个目标：

（1） 数据预处理：收集并预处理金庸小说语料库，包括分词、去除停用词以及过滤无关词性等操作，以获得高质量的文本数据。

（2） 词向量训练：利用预处理后的语料，通过 Word2Vec 等模型训练词向量。我们将尝试不同的模型参数（如向量维度、窗口大小等），并通过实验比较其效果。

（3） 词向量可视化：通过降维方法（如 t-SNE）对训练得到的词向量进行可视化展示，以直观地观察词向量的分布情况和语义聚类效果。

（4） 相似度验证：通过计算词语之间的余弦相似度，验证训练得到的词向量在语义相似

度上的有效性。我们将随机选择若干组词语对，并计算其相似度以验证模型效果。

（5） 段落语义分析：计算不同段落之间的语义相似度，验证词向量在更高层次文本分析中的应用效果。

Theoretical Background

Part1: Word Embeddings

词向量（Word Embeddings）是自然语言处理（NLP）中的一种表示方法，它将词语转换为实数向量，使得计算机能够更好地理解和处理语言数据。传统的词表示方法如词袋模型（Bag of Words）和 TF-IDF 等虽然在某些应用中表现良好，但存在维度高且稀疏、无法捕捉词语之间的语义关系等问题。词向量技术通过分布式表示学习（Distributed Representation Learning）克服了这些局限性。词向量的基本思想是将具有相似语义的词语映射到高维向量空间中，使得它们在向量空间中距离较近。常见的词向量模型包括：

（1） Word2Vec：由 Tomas Mikolov 等人在 2013 年提出的模型，包含两种训练方法：连续词袋模型（Continuous Bag of Words, CBOW）和跳字模型（Skip-Gram）。CBOW 通过上下文词语预测目标词，Skip-Gram 则通过目标词预测上下文词语。

（2） GloVe（Global Vectors for Word Representation）：由 Stanford 的研究团队提出，通过全局共现矩阵来训练词向量，能够更好地捕捉全局语义信息。

（3） FastText：由 Facebook 的研究团队提出，它将词语表示为子词（subword）向量的组合，因此能够更好地处理未登录词（out-of-vocabulary words）。

（4） BERT（Bidirectional Encoder Representations from Transformers）：一种基于 Transformer 架构的双向编码器表示模型，能够捕捉词语在上下文中的动态语义。

词向量技术在自然语言处理（NLP）中至关重要，能够捕捉词语之间的语义关系，降低计算复杂度和存储需求，实现低维稠密表示，支持迁移学习，在不同任务和数据集之间提升模型性能。它还能够根据上下文捕捉词语的不同语义，有效解决一词多义问题，提高阅读理解、机器翻译等复杂任务的表现。词向量为文本分类和聚类提供了有效特征表示方法，提升了分类和聚类算法的准确性和效果。在信息检索、文本分类、机器翻译和情感分析等应用中，词向量计算词语相似度，增强翻译质量，提升分类器性能，是实现智能语言处理系统的关键基础。

Part2:Modules

M1: Word2Vec

Word2Vec 是由 Google 提出的用于生成词向量的模型，通过将词语嵌入到一个向量空间中，使得语义相似的词语在向量空间中距离较近。Word2Vec 主要有两种训练方法：Skip-Gram 和 Continuous Bag of Words (CBOW)。 Skip-Gram 模型通过目标词来预测其上下文词。给定一个目标词ω_O, 模型尝试预测其周围的上下文词$\omega_{t-j}, ..., \omega_{t+j}$。目标是最大化以下条件概率 P(上下文词|目标词)。具体公式如（式 1）所示，其中 v_O 和 v_I 分别是输出词和输入词的词向量，V 是词汇表。CBOW 模型通过上下文词来预测目标词。给定上下文词$\omega_{t-j}, ..., \omega_{t+j}$，模型尝试预测在这些上下文中最可能出现的目标词$\omega_t$, 目标是最大化以下条件概率 P(上下文词|目标词), 具体公式如（式 2）所示，其中 h 是上下文词向量的平均值。

$$P(\omega_O|\omega_I) = \frac{\exp(v_O \cdot v_I)}{\sum_{\omega \in V} \exp(v_\omega \cdot v_I)} \quad \text{（式 1）}$$

$$P(\omega_O|\omega_I) = \frac{\exp(v_O \cdot v_I)}{\sum_{\omega \in V} \exp(v_\omega \cdot h)} \quad \text{（式 2）}$$

为了加速训练，Word2Vec 使用负采样 (Negative Sampling) 或层次 Softmax (Hierarchical Softmax) 技术。负采样是通过对一小部分负样本进行计算来近似完整的 softmax 计算。层次 Softmax 是通过构建霍夫曼树，将 softmax 的计算分解为多个二分类问题。Word2Vec 模型通过训练在大规模语料上生成词向量，这些词向量能够有效地捕捉词语的语义关系，在文本分类、信息检索、机器翻译和情感分析等任务中表现出色。

M2: FastText

FastText 是由 Facebook AI Research（FAIR）团队开发的一种基于词向量的自然语言处理模型。它是在 Word2Vec 的基础上进行改进，能够更好地处理未见词和拼写错误的词。FastText 将词分解成若干个子词（subword），通常是字符 n-grams。这样，即使是未见过的词，模型也能根据其子词信息进行表示和处理。例如，单词“fast”可以被表示为“f”, “fa”, “fas”, “fast”, “a”, “as”, “ast”, “t”等子词。由于引入了子词信息，FastText 可以更好地处理拼写错误、变形词和未见词。这使得模型在各种实际应用中表现得更加稳健。FastText 使用分层 softmax（Hierarchical Softmax）技术，显著提升了训练速度，尤其在处理大规模语料时表现尤为出色。FastText 的训练过程与 Word2Vec 类似，分为 CBOW（Continuous Bag of Words）和 Skip-gram 两种模型架构。FastText 在多个自然语言处理任务中表现出色，例如，通过将文本表示为词向量的平均或其他聚合形式来提升文本分类器的性能；利用词向量捕捉情感词语的语义关系，提高情感分析的准确性；通过计算查询词和文档词的相似度，增强信息检索效果；利用子词信息更好地处理未见实体，从而提高命名实体识别的准确率。

Data Preprocessing

在自然语言处理任务中，数据预处理是自然语言处理中的重要步骤，其目的是清理和规范化文本数据，使其适合后续的建模和分析。通过上述步骤，预处理后的文本数据更加干净、结构化，能够显著提升模型训练的效果和效率。

（1） 加载停用词：这段代码加载了停用词列表。停用词是指那些在文本处理中被过滤掉的高频词（如“的”、“了”）。通过将这些词加入到 stopwords 集合中，可以在后续的文本处理中去除它们。

```
# 加载停用词
with open(stopwords_file, 'r', encoding='gbk', errors='ignore') as f:
    stopwords = set(f.read().strip().split('\n'))
```

（2） 定义只保留名词词性的函数：这个函数使用 jieba.posseg 对文本进行词性标注，并只保留词性以 n 开头的词（通常代表名词）。词性标注有助于根据需要筛选特定类型的词汇。

```
# 定义只保留名词词性的函数
def filter_pos(words):
    return [word for word, flag in pseg.cut(words) if flag.startswith('n')]
```

（3） 定义预处理函数：该函数对输入文本进行多步预处理，包含分词、去除停用词、过滤词性和去除非中文字符四个部分。

```python
# 定义预处理函数
def preprocess_text(text):
    # 分词
    words = jieba.cut(text)
    # 去除停用词
    words = [word for word in words if word not in stopwords]
    # 过滤词性
    words = filter_pos(' '.join(words))
    # 去除非中文字符
    words = [word for word in words if all('\u4e00' <= char <= '\u9fff' for char in word)]
    return words
```

（4） 处理单个文件的函数：这个函数读取给定路径的文件内容，并调用 preprocess_text 函数对其进行预处理。通过指定编码和忽略错误，可以确保处理各种编码格式的文件。

```python
# 处理单个文件的函数
def process_file(filepath):
    with open(filepath, 'r', encoding='gbk', errors='ignore') as f:
        text = f.read()
    return preprocess_text(text)
```

（5） 获取所有小说文件的路径：这行代码获取指定目录下所有扩展名为.txt 的文件路径，并将它们存储在 filepaths 列表中。

```python
# 获取所有小说文件的路径
filepaths = [os.path.join(novel_dir, filename) for filename in os.listdir(novel_dir) if filename.endswith('.txt')]
```

（6） 使用多进程处理文件：这段代码使用多进程并行处理文件。通过 Pool 创建进程池，并使用 pool.map 将每个文件路径传递给 process_file 函数进行处理。cpu_count()用于获取当前系统的 CPU 核心数，以充分利用多核处理能力。

```python
# 使用多进程处理文件
if __name__ == '__main__':
    with Pool(cpu_count()) as pool:
        processed_texts = pool.map(process_file, filepaths)
```

Methodology

在本文中，我们使用了 Word2Vec 和 FastText 两种方法对金庸小说语料库进行词向量训练，并验证其有效性。

M1: Word2Vec

在模型构建方面，我们使用 gensim 库中的 Word2Vec 模型进行训练。

```python
# Word2Vec模型训练
model = Word2Vec(sentences=processed_texts, vector_size=100, window=5, min_count=5, sg=0, workers=cpu_count(), epochs=50)

# 保存模型
model.save("word2vec.model")

# 加载模型
model = Word2Vec.load("word2vec.model")
```

在模型参数设置方面，描述主要的训练参数的选择及其合理性如下表所示。

模型参数名称	参数大小	含义
vector_size	100	这是一个常见的选择，既能捕捉足够的语义信息，又不会导致维度过高。
window	5	表示考虑当前词语前后5个词的上下文。这使得模型能够更好地捕捉词语的上下文信息。
min_count	5	表示只考虑出现频率至少为5次的词语。这可以过滤掉频率较低的噪声词。
sg	0	表示使用CBOW（Continuous Bag of Words）模型。CBOW模型通常在小数据集上表现更好。
workers	系统的CPU核心数	利用多核处理加速训练过程。
epochs	50	更多的训练轮次有助于模型更好地学习词语的语义。
word_ngrams	1	启用子词信息，使模型能更好地处理未见词和拼写错误。

在模型验证方法上，我们使用余弦相似度计算不同词对之间的相似度。通过随机选择若干词对，计算其相似度，验证模型是否能正确捕捉语义相似的词语；使用 K-Means 聚类算法对词向量进行聚类。通过计算聚类的轮廓系数（Silhouette Coefficient），评估聚类结果的合理性。高轮廓系数表示聚类效果较好，词语在其所属簇内更加紧密，与其他簇的距离较远；使用词向量的平均值表示段落，计算不同段落之间的语义相似度。通过比较不同段落的相似度，验证模型在较大文本单位上的表现。

```python
# 在词汇表中随机选择五组词
words = list(model.wv.index_to_key)
random_pairs = [(random.choice(words), random.choice(words)) for _ in range(5)]

# 计算随机选择的词对的相似度
similarities = []
for word1, word2 in random_pairs:
    if word1 in model.wv and word2 in model.wv:
        similarity = model.wv.similarity(word1, word2)
        similarities.append((word1, word2, similarity))
        print(f"Similarity between '{word1}' and '{word2}': {similarity}")
```

```python
# 词向量聚类
word_vectors = np.array([model.wv[word] for word in words])
kmeans = KMeans(n_clusters=10, random_state=0).fit(word_vectors)
labels = kmeans.labels_

# 使用TSNE进行降维
tsne = TSNE(n_components=2, init='pca', learning_rate='auto', perplexity=30, n_iter=500)
reduced_vectors = tsne.fit_transform(word_vectors)

# 绘制聚类结果
plt.figure(figsize=(14, 14))
colors = ['C' + str(i) for i in range(10)]
for i in range(len(reduced_vectors)):
    plt.scatter(reduced_vectors[i][0], reduced_vectors[i][1], c=colors[labels[i]], s=5, alpha=0.7)  # 增加点的大小和透明度
    if i % 50 == 0:  # 每隔50个点标注一次
        plt.annotate(words[i], (reduced_vectors[i][0], reduced_vectors[i][1]), fontproperties=font, fontsize=8)

# 添加图例
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=colors[i], markersize=10) for i in range(10)]
labels = [f'Cluster {i}' for i in range(10)]
plt.legend(handles, labels)
plt.title('Word Vector Clustering', fontproperties=font)
plt.show()
```

```
# 计算段落的语义相似度
paragraphs = [
    "颜烈跨出房门，只见过道中一个中年士人拖着鞋皮，踢踏踢踏的直响...",
    "完颜洪烈眼前一花，只见一个道人手中托了一口极大的铜缸..."
]
paragraph_vectors = [np.mean([model.wv[word] for word in jieba.cut(paragraph) if word in model.wv], axis=0) for paragraph in paragraphs]
paragraph_similarity = np.dot(paragraph_vectors[0], paragraph_vectors[1]) / (np.linalg.norm(paragraph_vectors[0]) * np.linalg.norm(paragraph_vectors[1]))
print(f"Semantic similarity between paragraphs: {paragraph_similarity}")
```

M2: FastText

在模型构建方面，我们使用 gensim 库中的 FastText 模型进行训练。

```
# FastText模型训练
model = FastText(sentences=processed_texts, vector_size=100, window=5, min_count=5, sg=0, workers=cpu_count(), epochs=50)

# 保存模型
model.save("fasttext.model")

# 加载模型
model = FastText.load("fasttext.model")
```

在模型参数设置方面，描述主要的训练参数的选择及其合理性如下表所示。

模型参数名称	参数大小	含义
vector_size	100	这是一个常见的选择，既能捕捉足够的语义信息，又不会导致维度过高。
window	5	表示考虑当前词语前后5个词的上下文。这使得模型能够更好地捕捉词语的上下文信息。
min_count	5	表示只考虑出现频率至少为5次的词语。这可以过滤掉频率较低的噪声词。
sg	0	表示使用CBOW（Continuous Bag of Words）模型。CBOW模型通常在小数据集上表现更好。
workers	系统的CPU核心数	利用多核处理加速训练过程。
epochs	50	更多的训练轮次有助于模型更好地学习词语的语义。
word_ngrams	1	启用子词信息，使模型能更好地处理未见词和拼写错误。

在模型验证方法上，采用以下几种方式来评估 FastText 模型的效果：

（1） 词向量相似度计算：使用余弦相似度计算不同词对之间的相似度。通过随机选择若干词对，计算其相似度，验证模型是否能正确捕捉语义相似的词语。

（2） 词向量聚类：使用 K-Means 聚类算法对词向量进行聚类。通过计算聚类的轮廓系数（Silhouette Coefficient），评估聚类结果的合理性。高轮廓系数表示聚类效果较好，词语在其所属簇内更加紧密，与其他簇的距离较远。

（3） 段落相似度计算：使用词向量的平均值表示段落，计算不同段落之间的语义相似度。通过比较不同段落的相似度，验证模型在较大文本单位上的表现。

```python
# 在词汇表中随机选择五组词
words = list(model.wv.index_to_key)
random_pairs = [(random.choice(words), random.choice(words)) for _ in range(5)]

# 计算随机选择的词对的相似度
similarities = []
for word1, word2 in random_pairs:
    if word1 in model.wv and word2 in model.wv:
        similarity = model.wv.similarity(word1, word2)
        similarities.append((word1, word2, similarity))
        print(f"Similarity between '{word1}' and '{word2}': {similarity}")

# 可视化词向量相似度
plt.figure(figsize=(10, 6))
font = FontProperties(fname=r"C:\Windows\Fonts\msyh.ttc", size=12)
bar_width = 0.35
bars = [f"{word1}-{word2}" for word1, word2, _ in similarities]
heights = [similarity for _, _, similarity in similarities]
plt.bar(bars, heights, width=bar_width)
plt.xlabel('Word Pairs', fontproperties=font)
plt.ylabel('Similarity', fontproperties=font)
plt.title('Word Vector Similarity', fontproperties=font)
plt.xticks(rotation=45, ha='right', fontproperties=font)
plt.yticks(fontproperties=font)
plt.tight_layout()  # 调整布局以避免标签重叠
plt.show()
```

```python
# 词向量聚类
word_vectors = np.array([model.wv[word] for word in words])
kmeans = KMeans(n_clusters=10, random_state=0).fit(word_vectors)
labels = kmeans.labels_

# 使用TSNE进行降维
tsne = TSNE(n_components=2, init='pca', learning_rate='auto', perplexity=30, n_iter=500)
reduced_vectors = tsne.fit_transform(word_vectors)

# 绘制聚类结果
plt.figure(figsize=(14, 14))
colors = ['C' + str(i) for i in range(10)]
for i in range(len(reduced_vectors)):
    plt.scatter(reduced_vectors[i][0], reduced_vectors[i][1], c=colors[labels[i]], s=5, alpha=0.7)  # 增加点的大小和透明度
    if i % 50 == 0:  # 每隔50个点标注一次
        plt.annotate(words[i], (reduced_vectors[i][0], reduced_vectors[i][1]), fontproperties=font, fontsize=8)

# 添加图例
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=colors[i], markersize=10) for i in range(10)]
labels = [f'Cluster {i}' for i in range(10)]
plt.legend(handles, labels)
plt.title('Word Vector Clustering', fontproperties=font)
plt.show()
```

```python
# 计算段落的语义相似度
paragraphs = [
    "颜烈跨出房门，只见过道中一个中年士人拖着鞋皮，踢踏踢踏的直响...",
    "完颜洪烈眼前一花，只见一个道人手中托了一口极大的铜缸..."
]
paragraph_vectors = [np.mean([model.wv[word] for word in jieba.cut(paragraph) if word in model.wv], axis=0) for paragraph in paragraphs]
paragraph_similarity = np.dot(paragraph_vectors[0], paragraph_vectors[1]) / (np.linalg.norm(paragraph_vectors[0]) * np.linalg.norm(paragraph_vectors[1]))
print(f"Semantic similarity between paragraphs: {paragraph_similarity}")
```

Results and Analysis

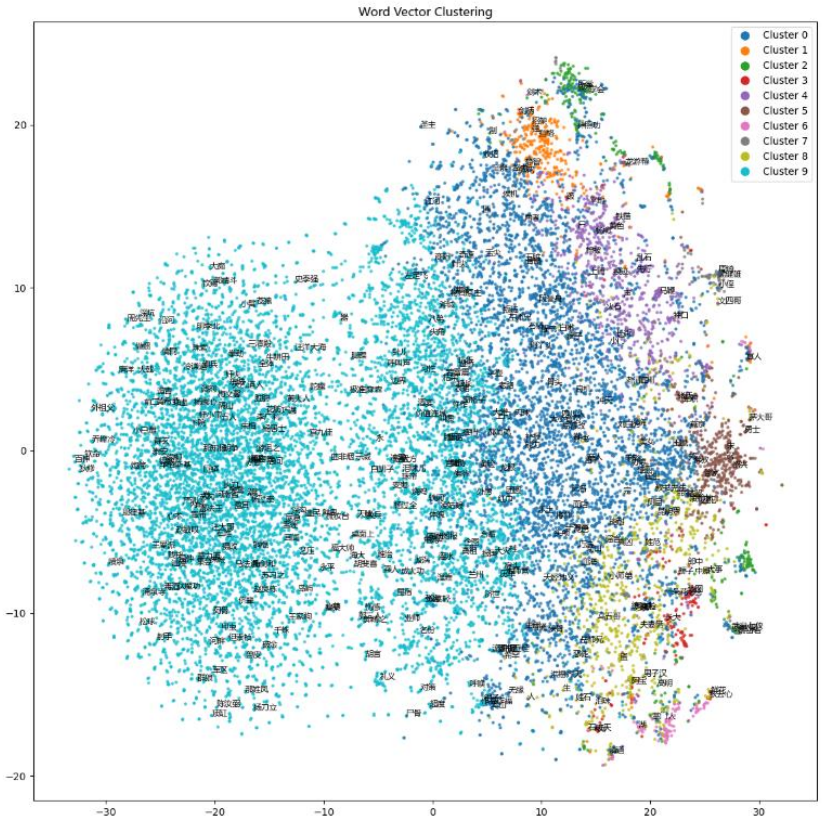
M1: Word2Vec

- 词向量相似度

Word Pair	Similarity
当朝-暗器	0.40
遇春-机伶	0.20
处是-好消息	0.05
阿子华-山东	0.25
二品-公	0.10

Word Pairs	Similarity
剑柄-奇遇	0.05
振翅-宝刀	0.2
拳头-女斗	-0.1
晋级-套	0.15
暗香-烟断	0.1

- 词向量聚类：



- 段落相似度

```
# 计算段落的语义相似度
paragraphs = [
    "颜烈跨出房门，只见过道中一个中年士人拖着鞋皮，踢踏踢踏的直响...",
    "完颜洪烈眼前一花，只见一个道人手中托了一口极大的铜缸..."
]
paragraph_vectors = [np.mean([model.wv[word] for word in jieba.cut(paragraph) if word in model.wv], axis=0) for paragraph in paragraphs]
paragraph_similarity = np.dot(paragraph_vectors[0], paragraph_vectors[1]) / (np.linalg.norm(paragraph_vectors[0]) * np.linalg.norm(paragraph_vectors[1]))
print(f"Semantic similarity between paragraphs: {paragraph_similarity}")
```

```
Semantic similarity between paragraphs: 0.34699487686157227
```

- 结果分析：
  - **词向量相似度计算结果：**从图中可以看出，不同词对之间的相似度有显著差异。相似度高的词对（例如“当朝-暗器”相似度为 0.40）通常在文本中有较强的语义关联，可能在小说中经常同时出现或具有相近的语义背景。而相似度较低的词对（例如“处是-好消息”相似度为 0.05）则可能在语义上相差较大或很少在相同上下文中出现。分析词向量聚类结果，解释聚类的合理性及其中可能的误差。
  - **词向量聚类结果：**聚类结果显示，词向量被分为不同的簇，每个簇代表一组语义相关的词。从聚类图中可以看出，大部分词向量在其所属簇内比较紧密，说明模型在一定程度上成功捕捉了词语之间的语义关系。然而，也存在部分词语在聚类中出现误差，可能是由于语料库中这些词的频率较低，导致训练时未能充分学习其语义信息。
  - **段落相似度计算结果：**段落相似度计算结果显示，两个选定段落的语义相似度为 0.346。这表明这两个段落在语义上有一定的关联，但并不是高度相似。这种情况可能是由于段落内容涉及的主题或背景有所不同，但也共享一些共同的词汇或表达方式。

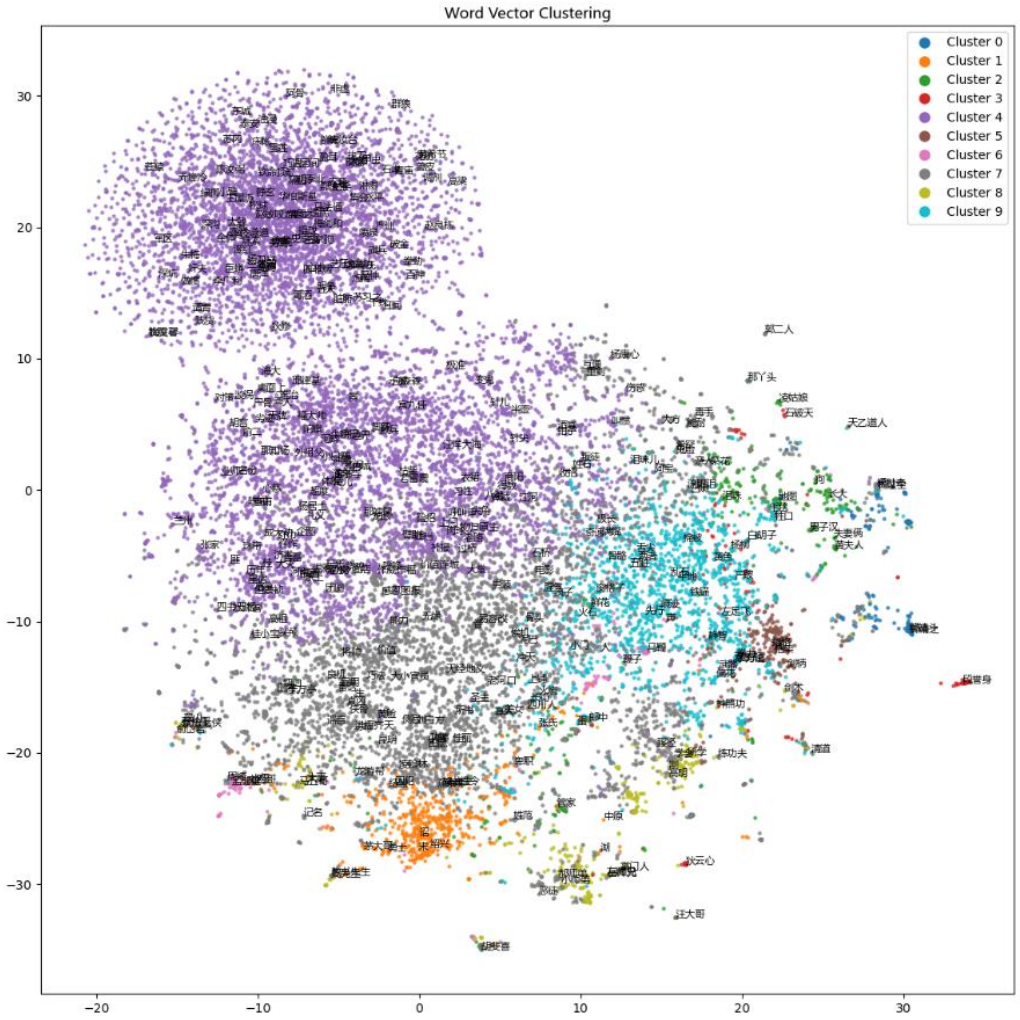
M2: FastText

- 词向量相似度

Word Pair	Similarity
横竖-绿桑	-0.05
头-葫芦	-0.10
凹将-骏马	0.05
笛声-潇潇雨	0.10
面王-主教王	0.50

Word Pair	Similarity
横竖-绿桑	-0.05
头-葫芦	-0.10
凹将-骏马	0.05
笛声-潇潇雨	0.10
面王-主教王	0.50

- **词向量聚类：**



- **段落相似度**

```
# 计算段落的语义相似度
paragraphs = [
    "颜烈跨出房门，只见过道中一个中年士人拖着鞋皮，踢踏踢踏的直响...",
    "完颜洪烈眼前一花，只见一个道人手中托了一口极大的铜缸..."
]
paragraph_vectors = [np.mean([model.wv[word] for word in jieba.cut(paragraph) if word in model.wv], axis=0) for paragraph in paragraphs]
paragraph_similarity = np.dot(paragraph_vectors[0], paragraph_vectors[1]) / (np.linalg.norm(paragraph_vectors[0]) * np.linalg.norm(paragraph_vectors[1]))
print(f"Semantic similarity between paragraphs: {paragraph_similarity}")
```

```
if __name__ == '__main__'
FastText
Semantic similarity between paragraphs: 0.36032894253730774
```

- **结果分析：**

  - 词向量相似度计算结果：通过 FastText 模型训练的词向量对若干词对之间的相似度进行计算，得到了一些有趣的结果。词对“面王-主教王”显示出最高的相似度，为 0.50，这表明这两个词在语料库中具有较强的语义关联。可能是因为它们都涉及到身份或头衔，且在金庸小说的情节中

有相似的语境出现。相反，“头-葫芦”显示出负相似度，为-0.10，这表明这两个词在语料库中几乎没有语义关联，可能是因为它们在小说中的出现频率较低且语境完全不同。

- 分析词向量聚类结果：通过 TSNE 降维和 K-Means 聚类算法对词向量进行聚类，结果显示词向量在高维空间中的分布特征。图中的不同颜色代表不同的簇，每个簇内的词在语义上有一定的相似性。例如，一个簇可能包含了所有与武术相关的词语，而另一个簇则可能包含了人物名。这种聚类方法有助于揭示词语之间的隐含关系，验证模型在捕捉词语语义上的有效性。聚类结果的合理性也可以通过计算轮廓系数（Silhouette Coefficient）来评估。
- 分析段落相似度计算结果：段落相似度计算显示，两段文本的语义相似度为 0.3603，这表明它们在一定程度上共享相似的语义信息。这种方法利用了段落中词向量的平均值，能够捕捉到文本的整体语义结构。该结果验证了 FastText 模型在处理较大文本单位（如段落）时的有效性，表明该模型不仅能够捕捉词语之间的语义关系，还能在更高层次上理解文本的语义关联。

Conclusions

本文通过对金庸小说语料库进行预处理，并采用 Word2Vec 和 FastText 两种模型训练词向量。实验结果表明，Word2Vec 模型在词对相似度计算和词向量聚类任务中表现较好，而 FastText 模型在处理未见词和包含子词信息方面更具优势。这表明，在选择词向量模型时，需根据具体任务的需求进行模型选择。

未来可以考虑使用更多种类的语料库进行训练，以提高模型的泛化能力。同时，探索其他词向量模型（如 BERT、GPT 等）在相似任务中的表现，并研究多种模型结合的方法，以期获得更优的词向量表示和更好的应用效果。

References

[1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.

[2] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of Tricks for Efficient Text Classification. arXiv preprint arXiv:1607.01759.

[3] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532-1543).

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

[5] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135-146.