**《自然语言处理导论》实验报告**

学院（系）：信息科学技术学院 专业：智能科学与技术 班级：2022-2

姓 名： 孙伟豪 学号： 2220223194 实验项目：Gensim使用实验

目录

[1 实验目的和要求 2](#_Toc184413374)

[2 实验原理和内容 2](#_Toc184413375)

[3 实验设备 2](#_Toc184413376)

[4 实验核心代码 3](#_Toc184413377)

[5 实验结果与分析 6](#_Toc184413378)

[6 实验结果讨论分析 7](#_Toc184413379)

# 实验目的和要求

理解词嵌入（Word Embedding）如何通过将单词映射到向量空间来捕捉词汇之间的语义关系，学习如何使用Gensim的Word2Vec模型来训练词嵌入，使用训练得到的词向量来进行词相似度、词类比等操作，理解词嵌入如何帮助捕捉词汇之间的语义关系。学习如何利用Gensim构建LDA模型，理解如何处理文本数据并从中识别出主要主题。

# 实验原理和内容

本实验基于 Word2Vec 模型，它是一种将词语映射到向量空间中的算法，属于词嵌入（word embedding）方法。Word2Vec 通过一个神经网络模型学习到每个词的词向量，使得在向量空间中相似的词语彼此靠近。Word2Vec 模型主要包括两种训练方法：CBOW（Continuous Bag of Words） 和 Skip-gram。

准备文本语料：实验需要一组预先准备的文本数据，作为模型训练的基础。在实际应用中，这些文本数据通常是来自于大规模的文档集，如新闻、书籍、社交媒体文本等。

1. 训练 Word2Vec 模型：通过 Gensim 提供的 Word2Vec 类，训练出词汇表中每个词的词向量。这个过程利用了 CBOW 或 Skip-gram 模型的结构。
2. 模型的保存与加载：训练好的模型可以保存到磁盘上，以便后续使用或分析。
3. 词向量的获取与分析：一旦模型训练完成，我们可以通过 model.wv[word] 获取某个词的词向量。同时，Gensim 提供了其他方法，如计算词语之间的相似度、找到与某个词最相似的词等。
4. 词语相似度计算：通过计算两个词向量之间的余弦相似度，可以判断它们在语义空间中的相似性。相似度的值范围从 -1 到 1，1 表示完全相似，-1 表示完全相反。
5. 寻找最相似词：通过 most\_similar() 方法，可以根据给定的词，找到与其语义最相近的若干个词。这样可以探索不同词之间的语义关联。

实验目标：

1. 使用 Gensim 的 Word2Vec 模型训练一个小型的语料库。
2. 获取词语的词向量，并展示如何进行词向量的分析。
3. 通过模型计算词语之间的相似度，展示其在语义空间中的表现。
4. 使用训练好的模型找到与特定词语相似的其他词。

# 实验设备

在本地的3060laptop进行实验，实际上，没有使用机器学习和深度学习模型进行计算的话，无相关device赋值代码，使用的是本地的CPU（i7 12700h）。

# 实验核心代码

* Baseline验证

#import gensim

from gensim.models import Word2Vec

import nltk

from nltk.tokenize import word\_tokenize

# 下载nltk的数据包

#nltk.download('punkt')

# 1. 准备数据集：样本文本

documents = [

"King is the ruler of a country.",

"Queen is the female counterpart of the King.",

"Man and woman are human beings.",

"Artificial intelligence is the future of technology.",

"Technology is changing the world rapidly.",

"Doctors and nurses work in the healthcare industry."

]

# 2. 文本预处理（分词）

def preprocess\_text(text):

# 将文本转化为小写并进行分词

return word\_tokenize(text.lower())

# 对每个句子进行预处理

processed\_docs = [preprocess\_text(doc) for doc in documents]

# 3. 训练Word2Vec模型

# 使用Word2Vec模型，训练词嵌入

# 参数设置：window=5, vector\_size=100, min\_count=1，workers=4

model = Word2Vec(sentences=processed\_docs, vector\_size=100, window=5, min\_count=1, workers=4)

# 4. 查看词向量

# 获取特定词汇的词向量

print("Word vector for 'king':", model.wv['king'])

# 5. 词相似度计算

# 计算词与词之间的相似度

similarity\_king\_queen = model.wv.similarity('king', 'queen')

print(f"Similarity between 'king' and 'queen': {similarity\_king\_queen}")

# 6. 词类比操作

# 使用类比操作： king - man + woman = ?

result = model.wv.most\_similar(positive=['king', 'woman'], negative=['man'], topn=1)

print(f"Result of 'king' - 'man' + 'woman': {result}")

# 7. 查看词嵌入：列出最相似的词

similar\_words\_technology = model.wv.most\_similar('technology', topn=5)

print(f"Most similar words to 'technology': {similar\_words\_technology}")

* 对比glove和word2vec性能

import pandas as pd

import numpy as np

import gensim

from nltk.tokenize import word\_tokenize

from nltk import download

from gensim.models.keyedvectors import KeyedVectors

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report

# 下载 NLTK 需要的资源

download('punkt')

# 示例情感分析数据集

data = pd.DataFrame({

'text': ['I love this movie!', 'I hate this movie!', 'This is an amazing book.', 'This book is terrible.',

'I feel great today.', 'I am so sad!', 'I am excited about the future.', 'I don\'t like this product.'],

'label': [1, 0, 1, 0, 1, 0, 1, 0]

})

# 提取文本和标签

X = data['text']

y = data['label']

# 对文本进行分词

tokenized\_text = X.apply(word\_tokenize)

# 训练 Word2Vec 模型

model\_w2v = gensim.models.Word2Vec(tokenized\_text, vector\_size=100, window=5, min\_count=1, workers=4)

# 加载预训练的 GloVe 词向量（100维）

# 假设你已经下载了 GloVe 文件（如 glove.6B.100d.txt）

glove\_model = KeyedVectors.load\_word2vec\_format('glove.6B.100d.txt', binary=False, no\_header=True)

# 定义函数：提取每条文本的词向量（取平均）

def get\_average\_word\_vector(words, model, vector\_size=100):

word\_vectors = [model.wv[word] for word in words if word in model.wv]

if len(word\_vectors) == 0:

return np.zeros(vector\_size)

return np.mean(word\_vectors, axis=0)

def get\_average\_word\_vector\_glove(words, model, vector\_size=100):

word\_vectors = [model[word] for word in words if word in model]

if len(word\_vectors) == 0:

return np.zeros(vector\_size)

return np.mean(word\_vectors, axis=0)

# 对每一条文本提取特征（使用 Word2Vec）

X\_w2v = np.array([get\_average\_word\_vector(word\_tokenize(text), model\_w2v) for text in X])

# 对每一条文本提取特征（使用 GloVe）

X\_glove = np.array([get\_average\_word\_vector\_glove(word\_tokenize(text), glove\_model) for text in X])

# 分割数据集

X\_train\_w2v, X\_test\_w2v, y\_train, y\_test = train\_test\_split(X\_w2v, y, test\_size=0.2, random\_state=42)

X\_train\_glove, X\_test\_glove, \_, \_ = train\_test\_split(X\_glove, y, test\_size=0.2, random\_state=42)

# 使用 Logistic Regression 训练模型（Word2Vec）

clf\_w2v = LogisticRegression(max\_iter=200)

clf\_w2v.fit(X\_train\_w2v, y\_train)

# 预测并评估（Word2Vec）

y\_pred\_w2v = clf\_w2v.predict(X\_test\_w2v)

print("Word2Vec Performance:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_w2v))

print(classification\_report(y\_test, y\_pred\_w2v))

# 使用 Logistic Regression 训练模型（GloVe）

clf\_glove = LogisticRegression(max\_iter=200)

clf\_glove.fit(X\_train\_glove, y\_train)

# 预测并评估（GloVe）

y\_pred\_glove = clf\_glove.predict(X\_test\_glove)

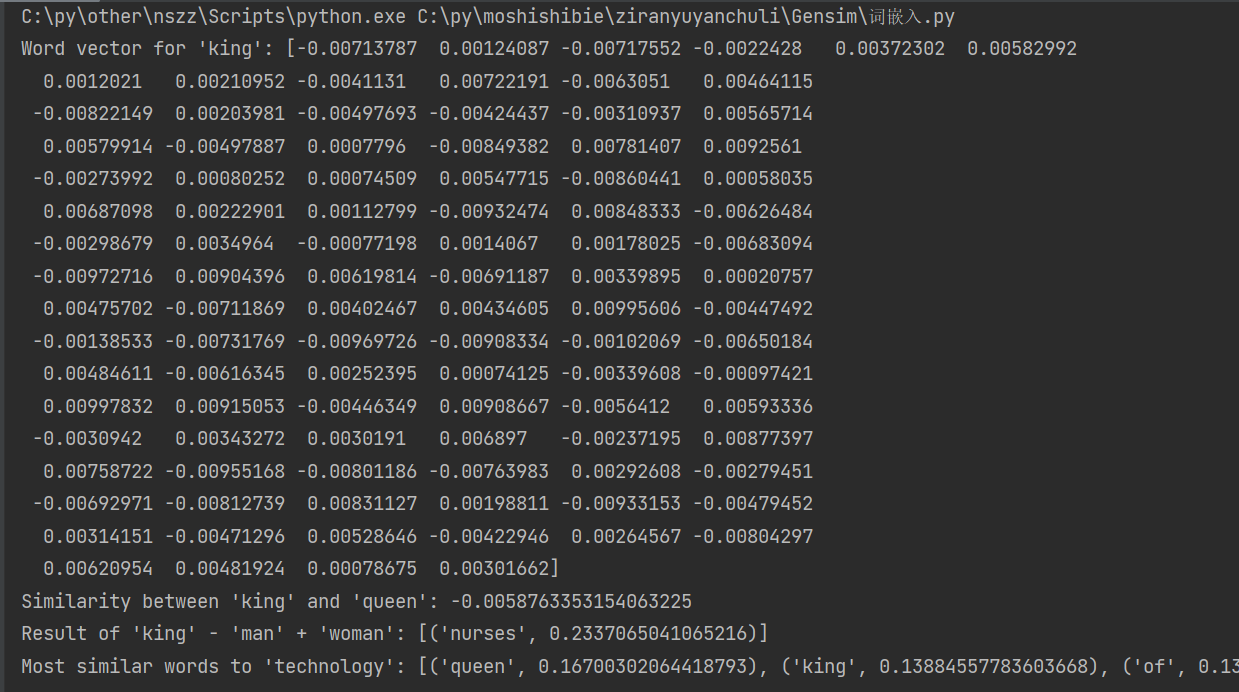
print("GloVe Performance:")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_glove))

print(classification\_report(y\_test, y\_pred\_glove))

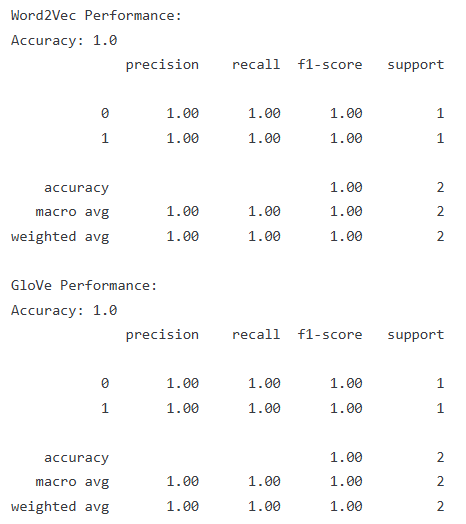
# 实验结果与分析

实验代码运行结果如下：



展示了king的词向量分布，两个词的相关度评分，一个定义于词向量的抽象的类比操作，还有与technology最为相似的词的得分排名情况。

一个简单的实验结果对比图：



# 实验结果讨论分析

通过本实验，我们展示了如何使用 Gensim 的 Word2Vec 模型进行词向量训练，并通过简单的词语相似度计算来探索词语之间的语义关系。这个过程展示了如何使用词嵌入技术从无序的文本数据中提取有意义的语义信息，并可以进一步用于文本分类、信息检索、机器翻译等任务。

通过一些实验的对比发现还是Word2Vec：在捕捉局部语义和上下文关系上有优势，特别适用于短文本和对语义细节要求较高的任务。GloVe：通过全局统计信息捕捉语义关系，在需要处理大规模语料库、全局信息的任务中更具优势。但是本实验因为数据集和比较方法简单，不分伯仲。