Project 1：词向量的训练

21307140069 田沐钊

Part 1：Word2Vec的原理

1.1 基本思路

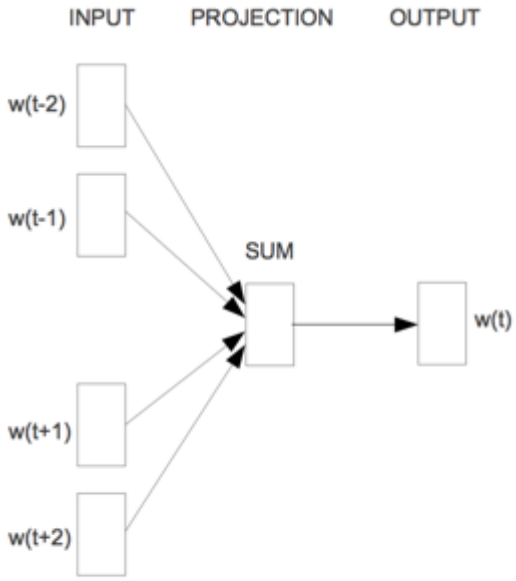
Word2Vec是一种用于将词语表示为连续向量的算法，其中`CBOW（Continuous Bag-of-Words）`和`Skip-gram`是两种常用的Word2Vec模型。其基本思路如下：

1. 准备一个足够长的可学习文本语料。
2. 将词汇表里的每个单词初始化为一个向量。
3. 对于文本中的每个位置t，称该位置的词为中心词c，而上下文的词语为o。
4. 我们希望能利用向量计算出的相似度来用c预测o或用o预测c。
5. 通过学习来优化向量，使预测的概率尽可能的大。

1.2 传统的 CBOW 和 Skip-gram 原理

CBOW模型的目标是通过给定上下文中的词语来预测目标词语。它的基本思想是将上下文中的词语向量进行平均，然后通过一个隐藏层将这些向量映射到目标词语的向量。具体步骤如下：

1. 输入层：将上下文中的词语向量（通常使用one-hot编码或其他向量表示方法）作为输入。
2. 隐藏层：将输入层的词语向量进行平均，然后将平均向量输入到隐藏层。隐藏层是一个全连接层，其权重矩阵表示词语的映射关系。
3. 输出层：隐藏层的输出通过一个线性变换和一个softmax函数，得到每个词语的概率分布。目标是使得目标词语的概率最大化。

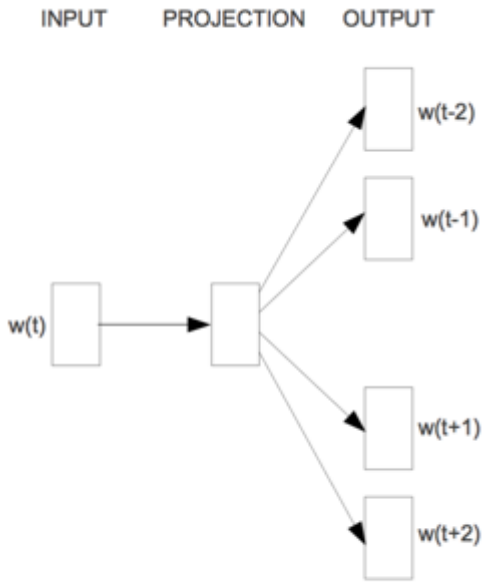


CBOW

训练CBOW模型的目标是最大化给定上下文预测目标词语的对数似然。通过反向传播算法，可以更新隐藏层和输出层的权重矩阵，以优化模型的性能。

与CBOW不同，**Skip-gram模型**的目标是通过给定目标词语来预测上下文词语。它的思想是使用目标词语来预测上下文词语，以捕捉词语之间的关系。具体步骤如下：

1. 输入层：将目标词语向量作为输入。
2. 隐藏层：将输入层的词语向量通过一个隐藏层映射到一个中间向量（通常是一个低维向量）。隐藏层的权重矩阵表示词语的映射关系。
3. 输出层：隐藏层的输出通过一个线性变换和一个softmax函数，得到每个上下文词语的概率分布。目标是使得给定目标词语的条件下，预测上下文词语的概率最大化。



Skip-gram

与CBOW模型类似，训练Skip-gram模型的目标也是最大化对数似然。通过反向传播算法，可以更新隐藏层和输出层的权重矩阵，以优化模型的性能。

以Skip-gram为例，其**似然函数和目标函数**如下：

$$L(\theta)=\prod_{t=1}^{T}\prod_{-m\leq j\leq m}P(w_{t+1}|w_t;\theta)(j\neq 0)$$

$$J(\theta)=-\frac{1}{T}logL(\theta)=-\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m}\log P(w_{t+1}|w_t;\theta)$$

1.3 基于负采样的优化

但是在使用以上的传统方法时，从隐藏层到输出的softmax层要计算所有词的softmax概率，计算量很大，故出现了分别基于**Hierarchical Softmax**和基于**Negative Sampling**的模型改进。下面主要讲解基于**负采样**的模型改进原理。

在基于负采样的CBOW和Skip-gram中，我们不再使用softmax函数来计算所有词语的概率分布，而是采用了二元逻辑回归的方法。具体来说，我们将目标词语与一些负样本进行对比，判断它们是否是上下文词语。

在基于负采样的CBOW模型中：

1. 输入层：将上下文中的词语向量作为输入。
2. 隐藏层：将输入层的词语向量进行平均，然后将平均向量输入到隐藏层。
3. 输出层：隐藏层的输出与目标词语的向量进行点积操作，然后通过sigmoid函数进行激活，输出一个概率值表示目标词语是否出现在上下文中。
4. 训练目标：最大化给定上下文预测目标词语的对数似然，并使用随机梯度下降等优化算法来更新模型参数。同时，模型会随机选择一些负样本（即未出现在上下文中的词语）作为对比，使得目标词语的概率被最大化，而负样本的概率被最小化。

在基于负采样的Skip-gram中：

1. 输入层：将目标词语向量作为输入。
2. 隐藏层：将输入层的词语向量通过一个隐藏层映射到一个中间向量。
3. 输出层：隐藏层的输出与上下文词语的向量进行点积操作，然后通过sigmoid函数进行激活，输出概率值表示上下文词语是否出现在目标词语周围。
4. 训练目标：最大化给定目标词语预测上下文词语的对数似然，并使用随机梯度下降等优化算法来更新模型参数。同时，模型会随机选择一些负样本作为对比，使得上下文词语的概率被最大化，而负样本的概率被最小化。

在逻辑回归中，我们的正例应该期望满足：

$$P(content(w_0),w_i)=\sigma(s_{w_0}^T\theta^{w_i}),y_i=1,i=0$$

我们的负例期望满足：

$$P(content(w_0),w_i)=1-\sigma(s_{w_0}^T\theta^{w_i}),y_i=0,i=1,2,\dots neg$$

我们期望可以最大化下式：

$$\prod_{i=0}^{neg} P(content(w_0), w_i) = \sigma(s_{w_0}^T \theta^{w_0}) \prod_{i=1}^{neg} (1 - \sigma(s_{w_0}^T \theta^{w_i}))$$

此时模型的似然函数为：

$$\prod_{i=1}^{neg} \sigma(s_{w_0}^T \theta^{w_0})^{y_i} (1 - \sigma(s_{w_0}^T \theta^{w_i}))^{1-y_i}$$

其负对数似然函数为：

$$L = -\sum_{i=0}^{neg} y_i \log(\sigma(x_{w_0}^T \theta^{w_i})) - (1 - y_i) \log(1 - \sigma(s_{w_0}^T \theta^{w_i}))$$

我们将其作为训练的目标函数。

Part 2：数据处理与分析

2.1 数据处理

首先，我们将.json文件中的数据读出为词典，并将所有条目的answer数据和title数据拼接在一起作为我们的训练数据，将其以字符串的形式存入列表中。

接下来使用jieba包对其进行分词操作，这里我们采用精确模式，将文本分成不重叠的词语，一句文本存入一个列表中，每个列表中的每个元素都是一个词语或标点。

其代码如下：

```
import json
import jieba

data = []
with open('baike_qa_train.json', 'r', encoding='utf-8') as f:
    for line in f.readlines():
        if line != '':
            sample = json.loads(line)
            data.append(sample.copy())

train_data = [i['answer'] for i in data] + [i['title'] for i in data]

seg_data = []
for i in train_data:
    seg_i = jieba.lcut(i)
    seg_data.append(seg_i)
```

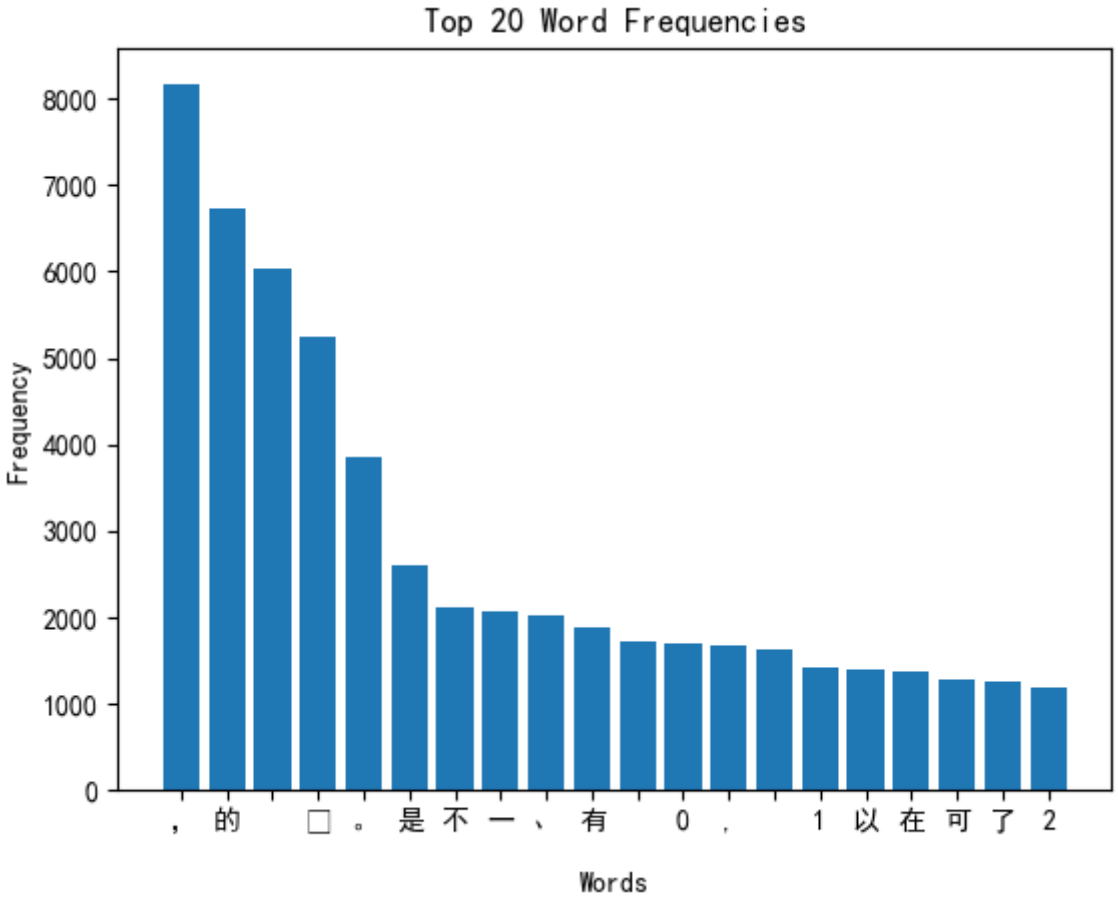
2.2 数据分析

为了更好地了解数据的特点和结构，从而可以针对性地进行数据处理以提高使用效率和训练效果，我们对数据进行了如下词频统计和可视化分析。

这里为了更方便快捷地进行统计，我们仅在所有数据中随机采样了1000条数据进行了统计，但仍具有代表性。

```
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud


# 统计词频
def count_word_frequency(data):
    word_counts = Counter()
    for i in data:
        word_counts.update(i)
    return word_counts

# 将最高频的二十个词的词频分布绘制成柱状图
def visualize_word_frequency(word_counts, num_words=20):
    words, counts = zip(*word_counts.most_common(num_words))
    print(words)
    plt.bar(words, counts)
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.title(f'Top {num_words} Word Frequencies')
    plt.show()

# 绘制词云图
def draw_word_cloud(data):
    wordcloud = WordCloud(width=800, height=400, font_path='msyh.ttc',
background_color='white', colormap='Blues_r').generate(' '.join(data))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.show()

word_counts = count_word_frequency(train_data)
visualize_word_frequency(word_counts)
draw_word_cloud(train_data)
```

我们得到高频词的柱状图统计和词云图如下（柱状图中横坐标上显示的方框与空白均为matplotlib不支持字体的乱码）：



容易发现在训练文本中高频出现的词，往往是一些缺乏实义的标点符号、连词、介词、冠词，故我们选择**对原数据中的高频数据进行降采样**，其目的如下：

1. **减少训练时间和内存消耗：**通过降低高频词语的采样概率，可以减少训练样本的数量，从而加快训练速度并降低内存消耗。
2. **平衡高频词语和低频词语的训练权重：**一些高频词语（如常见的功能词、停用词）往往没有很强的语义信息，而一些低频词语（如专业术语、稀有词汇）可能具有更丰富的语义信息。通过降低高频词语的采样概率，可以平衡高频和低频词语的训练权重，使得模型在学习词向量时更加关注那些更有信息量的低频词语。

3. **提升低频词语的学习效果：**低频词语出现的次数较少，因此它们在训练中的样本数量相对较少。如果不对高频词语进行降采样，由于高频词语的样本数量较多，模型可能更容易学习到高频词语的上下文信息，而对于低频词语的学习效果可能不理想。通过降低高频词语的采样概率，可以增加低频词语的样本比例，提升对低频词语的学习效果。

而具体的降采样操作将在创建模型的部分描述。

而对于非中文汉字的标点符号、乱码，我们直接进行清洗如下：

```
import re

def clean_non_chinese(list):
        cleaned_lst = []
        for word in list:
            # 使用正则表达式去除非中文字符
            word = re.sub(r'[^\u4e00-\u9fff]', '', word)
            if word:
                cleaned_lst.append(word)
        return cleaned_lst


for i in seg_data:
    i = clean_non_chinese(i)
```

Part 3：创建模型并训练

3.1 降采样模型的创建、实现与训练

以下为创建及训练模型的代码：

```
from gensim.models import Word2Vec
from collections import Counter

# 创建Word2Vec模型
model = Word2Vec(sentences=seg_data, min_count=1, vector_size=100, window=5,
workers=8)

# 将所有文本合并入一个列表，方便统计词频
nested_data = [i for sublist in tqdm(seg_data) for i in sublist]
frequency = Counter(nested_data)
# 记录最大词频
max_freq = frequency.most_common(1)[0][1]
for word, freq in tqdm(frequency.items()):
    # 对所有词语的采样概率进行重定义
    model.wv.set_vecattr(word, 'sample_probability', (1 - ((freq - 1) / max_freq))
** 20)

# 训练模型
model.train(seg_data, total_examples=len(seg_data), epochs=5)
```

```
model.save('downsampling.model')
```

再重定义的过程中，我们用

$$(1-((freq_i-1)/freq_{max}))^{20}$$

这个二次函数对采样概率进行赋值，从而保证词频越高的词采样概率越低，而词频越低的词采样概率越高。

我们对通过这种方式处理得到的采样概率进行部分打印展示如下（为了统计方便，这里的词频仅为从全部数据中随机采样了1000条中的统计结果）：

```
词语   词频   采样概率
地球:  4,  0.9908390595455422
上:   325,    0.36086596105230345
重力:  1,  1.0
作用:  62,     0.8286408816530948
一直:  33,     0.9062989545660832
是:   1614,   0.0034025700656500836
指向:  5,  0.9878031705064209
球心:  2,  0.9969374497537493
的:   6521,   5.172684568679965e-77
因此:  43,     0.8787707536771067
只要:  50,     0.8599756766518718
头:   13,     0.9638321898765299
远离:  5,  0.9878031705064209
人们:  34,     0.9035097009232543
就:   631,    0.13106793303196293
回:   11,     0.9697725852898148
感到:  7,  0.981757882344216
头朝:  1,  1.0
勤:   1,  1.0
洗澡:  7,  0.981757882344216
养成:  17,     0.9520549465120265
好:   288,    0.406491954906355
卫生习惯:    1,  1.0
通过:  57,     0.8415628974122735
没有:  222,    0.5017957571876629
```

可以看到其能有效降低某些缺乏足够实义的高频虚词的采样频率。

3.2 默认模型与其他模型的训练

除了该降采样的模型之外，我们通过修改超参数和训练方案准备了多个模型进行对比分析，其代码如下：

```
# 未经过降采样的默认配置下的模型
model = Word2Vec(sentences=seg_data, vector_size=100, window=5, min_count=1,
workers=8)
model.save('normal.model')

# 相对于normal，将窗口尺寸由5调大为10，其他配置不变，训练出的模型
```

```
model = Word2Vec(sentences=seg_data, vector_size=100, window=10, min_count=1,
workers=8)
model.save('bigger_window.model')

# 相对于normal，词向量尺寸由100降低为50，其他配置不变，训练出的模型
model = Word2Vec(sentences=seg_data, vector_size=50, window=5, min_count=1,
workers=8)
model.save('shorter_vector.model')

# 相对于normal，将训练方法由CBOW改为Skip-gram，其他配置不变，训练出的模型
model = Word2Vec(sentences=seg_data, vector_size=100, window=5, min_count=1,
workers=8, sg=1)
model.save('skip_gram.model')

# 相对于normal，将训练优化方法由负采样改为hierarchy softmax，其他配置不变，训练出的模型
model = Word2Vec(sentences=seg_data, vector_size=100, window=5, min_count=1,
workers=8, hs=1)
model.save('hierachy_softmax.model')

# 相对于normal，将迭代次数由5改到10，其他配置不变，训练出的模型
model = Word2Vec(sentences=seg_data, vector_size=100, window=5, min_count=1,
workers=8, epochs=10)
model.save('more_epochs.model')
```

Part 4：结果分析

4.1 对降采样模型训练结果的合理性分析

首先，我们针对降采样得到的模型进行各种性能分析如下：

```
model.wv.similarity('中国', '美国') # 计算两个词的余弦相似度
model.wv.most_similar('中国') # 取出与“中国”最相似的10个词
```

我们计算出 `中国` 与 `美国` 两个词的余弦相似度为：

```
0.7592292
```

此外，与 `中国` 最相似的10个词极其相似度为：

```
[('日本', 0.8104029893875122),
 ('美国', 0.7592291235923767),
 ('俄罗斯', 0.735672652721405),
 ('台湾', 0.7279495000839233),
 ('亚洲', 0.7138910889625549),
 ('我国', 0.6933271884918213),
 ('印度', 0.6747385859489441),
 ('外国', 0.6726107597351074),
 ('欧洲', 0.671537458896637),
 ('东亚', 0.665142834186554)]
```

可见 `中国` 与 `美国` 的相似度较高，并且与 `中国` 相似度较高的词语大多为知名的国家或地区名称，在直观上符合词语之间的相似度情况。可见**我们训练得到的词向量在一定程度上具有计算反映词语相似度的能力。**

此外，我们还做了如下计算：

```
model.wv.most_similar(positive=["中国", "纽约"], negative=["上海"]) # 获得 中国 - 上海 + 纽约 的词
```

```
[('美国', 0.7442100048065186),
 ('美国中央情报局', 0.6966733932495117),
 ('希拉里', 0.6900385022163391),
 ('英国', 0.6815199851989746),
 ('法兰西', 0.6553775072097778),
 ('拿破仑', 0.6528657674789429),
 ('荷兰人', 0.6513704061508179),
 ('原苏联', 0.6504176259040833),
 ('施瓦辛格', 0.6497381329536438),
 ('俄罗斯', 0.6482887864112854)]
```

我们使用`上海`和`中国`这两个词作为正向输入，`纽约`作为负向输入，通过运算得到一个新的词向量，并找出与该新向量最相似的词语。

结果显示，最相似的词语是`美国`，其相似度得分为0.7442。其他相似的词语包括`美国情报局`、`俄罗斯`、`英国`等等。这个结果符合课程中所讲授的词向量的线性性质，可以认为 `中国` 之于 `上海` 的关系类似于 `美国` 之于 `纽约` 的关系，而这种关系体现在词向量的线性性质上，可以认为 `中国 - 上海` 这个关系向量与 `美国 - 纽约` 这个关系向量高度相关或者平行。如果这两个关系向量平行，则容易得出 `中国 - 上海 + 纽约` 与 `美国` 对应的向量应该高度相关。而我们的计算结果也有力地说明了这一点。证明**我们训练得到的词向量具有良好的线性性质。**

接着，我们又计算了比 `中国` 更接近 `上海` 的词汇如下：

```
model.wv.closer_than("上海", "中国") # 比 中国 更接近 上海 的词汇
```

部分结果如下：

```
 '北京',
 '国内',
 '全国',
```

```
 '香港',
 '韩国',
 '广州',
 '深圳',
 '南京',
 '天津',
 '重庆',
 '成都',
 '广东',
 '杭州',
 '四川',
 '山东',
 '那边',
 '西安',
 '武汉',
 '浙江',
 '苏州',
 '青岛',
 '河南',
 '外地',
 '江苏',
 '济南',
...
```

可见比 `上海` 更接近 `中国` 的词也大多为中国的地区，符合一般的直观认知，说明我们训练得到的词向量不但能够在一定程度上反映词语之间的语义关联性，并且**通过这种关联性进行比较得出的结果也往往符合一般的语言认知。**

此外，我还对降采样得到的词向量进行了聚类处理，通过对不同聚类中的词语情况进行观察，我们可以对模型的语义分析能力进行一定的判断。其代码实现如下：

```python
from sklearn.cluster import KMeans
from tqdm import tqdm


word_vectors = model.wv.vectors

kmeans = KMeans(n_clusters=100)
clusters = kmeans.fit_predict(word_vectors)

# 将聚类结果存储在列表中
cluster_list = [[] for _ in range(100)]

for i, word in enumerate(model.wv.index_to_key):
    cluster_list[clusters[i]].append(word)

# 打印聚类
for i, cluster in enumerate(cluster_list):
    if i == 6:
        break
    print("Cluster", i+1, ":", cluster)
```

其部分结果展示如下：

```
Cluster 1 : ['王', '派', '今', '皇帝', '传', '将军', '三国', '姓', '曹操', '官', '李',
'孔子', '长安', '拜', '诸葛亮', '刘备', '遂', '陈', '百姓', '清朝', '明朝', '洛阳', '匈奴',
'弟子', '改名', '中原', '皇后', '刘', '唐朝', '大臣', '帝王', '墓', '氏', '杨', '死后', '出
身', '郡', '屈原', '吴', '秦', '赵', '史记', '秦始皇', '刘邦', '太子', '府', '关羽', '朝
廷', '黄帝', '项羽', '孟子', '南宋', '康熙', '东汉', '子孙', '任命', '诸侯', '大王', '吕布',
'蜀', '此人', '孙悟空', '自称', '天子', '元年', '女娲', '侯', '京城', '乾隆', '赵云', '大
唐', '殿' ...]
Cluster 2 : ['有', '多', '大', '一定', '高', '小', '比', '较', '非常', '不错', '无', '提
高', '快', '少', '明显', '长', '低', '最大', '保证', '很大', '越', '相对', '强', '不少',
'更好', '最高', '很快', '相当', '最佳', '差', '合适', '不够', '较大', '越来越', '足够', '很
少', '重', '有所', '短', '提升', '低于', '接近', '慢', '最低', '大于', '远', '巨大', '充
足', '高于', '有限', '很小', '轻', '显著', '极大', '弱', '加大', '而定', '不高', '更大', '很
强', '大大', '越高', '偏', '越大', '较长', '过大', '相差', '降', '高达', '太高', '远远', '太
大', '较强', '很长', '太小', '不太好', '好些', '多大', '不大', '超出', '节省', '落后', '较
差', '中等', '因人而异', '适中', '节约', '稳', '更高', '高出', '最有', '少于', '广', '不小',
'极少', '多一点', '多一些', '更佳', '太低', '偏高', '惊人', '多于', '超强', '相对来说', '总的
来说', '最具', '比起', '偏低', '同等', '不及', '很差', '太快', '速度慢', '极高', '较弱', '偏
向', '多些', '昂贵', '强劲', '大有', '得当', '太慢', '看重', '太少', '要少', '优于', '稀少',
...]
Cluster 3 : ['二', '三', '继续', '除', '一份', '定', '注', '现', '制', '凭', '八', '临
时', '除外', '急急', '视', '事先', '持', '附', '若干', '转入', '联', '到位', '是否是', '这
要', '估价', '暂', '如是', '急需', '另有', '备注', '追加', '一律', '一般来讲', '出入', '一
并', '此项', '任选', '批', '先行', '每位', '预', '就近', '每件', '符合要求', '五项', '事前',
'三方', '现时', '不计', '不定期', '给与', '留存', '免去', '即以', '变相', '两份', '并入',
'再作', '转运', '指以', '承接', '强烈要求', '已满', '指在', '仍为', '不设', '骗取', '撮合',
'仅限', '中所', '毛利', '车船', '伙食', '予', '费时', '我于', '内为', '微薄', '只限', '零
星', '自备', '先期', '中标', '安全措施', '期初', '需到', '按此', '劳力', '该法', '按计划',
'雇用', '空缺', '下表', '单上', '床位', '上班时间', '需经', '人行', '暂缓', '申明', '象征性',
'两方', '表上', '比照', '本票', '非专业', '中国建设银行'...]
Cluster 4 : ['含', '用于', '少量', '元素', '混合', '生成', '天然', '铁', '实验', '污染',
'化学', '物', '液体', '人工', '制成', '提取', '高温', '而成', '纯', '发酵', '分离', '过滤',
'浓', '铜', '储存', '有机', '品', '沉淀', '中性', '产物', '结晶', '复合', '有毒', '制品',
'甲醛', '净化', '耐', '低温', '回收', '固体', '质', '草酸', '吸附', '氨', '醋酸', '溶于',
'杂质', '无害', '铝', '浓缩', '渗透', '用作', '配制', '燃料', '基', '苯', '碱', '无色', '定
量', '碳', '溶剂', '添加剂', '饱和', '染料', '生化', '贮存', '无毒', '矿物', '柠檬酸', '内
含', '煤', '粉尘', '混合物', '中含', '汞', '化工', '一定量', '氟', '硫酸', '水解', '人造',
'放射性', '碳酸', '精制', '调制', '有机物', '硝酸', '硫', '重金属', '锰', '样品', '氯', '提
炼', '活性炭', '溶', '烟草', '镍', '澄清', '氧化物', '水溶性', '氢气', '液态', '无机', '污染
物', '可可', '萃取', '等量', '开采', '催化剂', '化学成分', '纯净水', '无味', '氮', '溶性',
'荧光', '尿素', '有色', '煤炭', '置换', '香精', '一氧化碳', '硅', '碳酸钙', '氧化剂', '熔化',
'酿造', '锡', '饮用水', '高浓度', '铀', '挥发性'...]
```

```
Cluster 5 : ['手', '头', '腹部', '左', '稍', '脚', '腿', '双手', '手指', '姿势', '右',
'上下', '按摩', '用力', '向上', '背', '头部', '两侧', '沿', '腰', '胸部', '大腿', '摩擦',
'足', '右侧', '耳朵', '向前', '四肢', '向下', '左侧', '颈部', '弯曲', '臀部', '小腿', '直',
'转动', '背部', '下肢', '腰部', '手臂', '右手', '左手', '两边', '行走', '一侧', '交替',
'颈', '站立', '内侧', '腿部', '肢体', '耳', '脖子', '弦', '侧面', '穴', '膝盖', '舌头', '下
垂', '双腿', '胸', '穴位', '食指', '蹲', '双脚', '吸气', '侧', '外侧', '两手', '双眼', '下
巴', '伸直', '斜', '拇指', '手掌', '肩膀', '额头', '捏', '小腹', '屁股', '头顶', '前方', '牵
引', '手腕', '肩', '横', '膝关节', '伸展', '腹肌', '按压', '摆动', '仰卧', '抬起', '眼皮',
'呼气', '下部', '双臂', '前端', '操', '中指', '倾斜', '脚趾', '负重', '直立', '筋', '枕',
'腋下', '上肢', '后背', '张开', '脚跟', '膝', '眼角', '腹', '胸前', '掌', '微微', '竖', '跳
动', '抬高', '手心', '肚脐', '脚尖', '无名指', '卧', '唇', '伸出', '轻柔', '两腿', '下颌',
'尾部', '胳膊', '向外', '哑铃', '肩部'...]
Cluster 6 : ['加入', '放入', '煮', '洗净', '盐', '油', '少许', '煎', '炒', '酱油', '味精',
'白糖', '切成', '倒入', '取出', '加水', '开水', '调', '碗', '葱', '姜', '汁', '淀粉', '烧',
'熟', '切', '搅拌', '面粉', '末', '锅', '冷却', '备用', '冰糖', '炖', '料酒', '丝', '粒',
'去皮', '花椒', '小火', '精盐', '白酒', '蒸', '调味', '汤匙', '蒜', '拌匀', '捞出', '煮熟',
'香油', '置', '拌', '煮沸', '制法', '粳米', '撒', '胡椒粉', '熬', '切片', '切碎', '沸水',
'炸', '锅内', '糯米', '渣', '黄酒', '大匙', '茶匙', '放进', '片刻', '火烧'...]
Cluster 7 : ['问题', '系统', '电脑', '游戏', '下载', '文件', '设置', '安装', '软件', '解
决', '网站', '手机', '操作', '程序', '用户', '网络', '运行', '升级', '病毒', '删除', '服务
器', '内存', '空间', '故障', '硬盘', '模式', '错误', '启动', '数据', '工具', '盘', '修改',
'更新', '显卡', '计算机', '版本', '连接', '上网', '驱动', '配置', '格式', '开机', '网页',
'主板', '清理', '文件夹', '关闭', '机器', '视频', '硬件', '重启', '上传', '浏览器', '操作系
统', '分区', '共享', '播放', '光盘', '注册表', '木马', '备份', '打印', '光驱', '杀毒软件',
'驱动程序', '死机', '打印机', '还原', '扫描', '重装', '主机', '默认', '补丁', '插件', '访问',
'客户端', '声卡', '魔兽', '卸载', '浏览', '冲突', '防火墙', '杀毒', '虚拟内存'...]
```

可以看到聚类分类的结果确实在相当程度上反映了不同词语的语义关联性。在分类结果中，聚类一的内容大多为中国古代历史的用语和人物，聚类二的内容大多为描述性的程度词，聚类三的内容多为事务办理用语，聚类四的内容大多为化学用语，聚类五的内容大多为身体部位，聚类六的内容大多为料理用语，聚类七的内容大多为电脑用语。不同聚类中的词语具有高度的语义一致性。而这些聚类本质上都是由词向量之间的余弦相似度进行分类的，由此可以证明我们证明的词向量**能够在相当大的范围上有效地反映出语义的关联性。**

综上，我们通过降采样训练得到的词向量各方面性能均表现良好，能够在不同程度上对语义进行有效表示。

4.2 对不同模型性能的比较分析

为了对不同模型训练得到的词向量进行可量化的评估，这里我们引入了额外的数据集以进行词向量的**词类比评估**和**词相似度评估**，其具体实现代码如下：

```python
def eval(model):
    analogy_path = 'eval/analogy.txt'
    pair_path = 'eval/pair.txt'
    analogy_scores = model.wv.evaluate_word_analogies(analogy_path)
    print('analogy_scores:', analogy_scores[0])
    similarity_scores = model.wv.evaluate_word_pairs(pair_path)
    print('similarity_scores:', similarity_scores)
```

词类比估计（Word Analogy）： 词类比估计任务旨在测试模型对于词语之间的类比关系的理解能力。例如，给定一个类比关系"A与B类似于C与？"，任务是找到一个词语D，使得A:B与C:D具有相似的关系。这种任务可以用来评估训练得到的词向量对于语义关联性的捕捉能力。具体的原理是，通过计算两个词向量的差异向量，然后寻找与差异向量最相似的词向量，找到与C:D类似的词语。通过这种方式，我们可以进行语义推理，例如"男人:女人与国王:？"，我们希望找到与"国王"在类似关系中的词语，即"皇后"。其得分**直接量化了其在测试数据中对类比性判断的准确率**，反映了词向量的语义表达能力。

词相似度估计（Word Similarity）： 词相似度估计任务旨在衡量两个词语之间的语义相似度或相关性。任务通常给出一对词语，然后要求模型给出它们之间的相似度得分。这种任务可以用来评估词向量在捕捉词语语义相似性方面的能力。具体的原理是，通过计算两个词向量之间的相似度或距离，然后将其映射到一个相似度得分。常用的相似度度量包括余弦相似度和欧氏距离。例如，对于词对("汽车", "自行车")，我们可以计算它们的词向量之间的相似度， 输出一个表示它们语义相似程度的得分。

其中词相似度估计的得分包括多个部分，具体形式如下：

```
similarity_scores: (PearsonRResult(statistic=0.4192044028992752,
pvalue=2.0282580055611441e-23), SignificanceResult(statistic=0.4549766571598661,
pvalue=8.850982592087141e-28), 3.7243947858473)
```

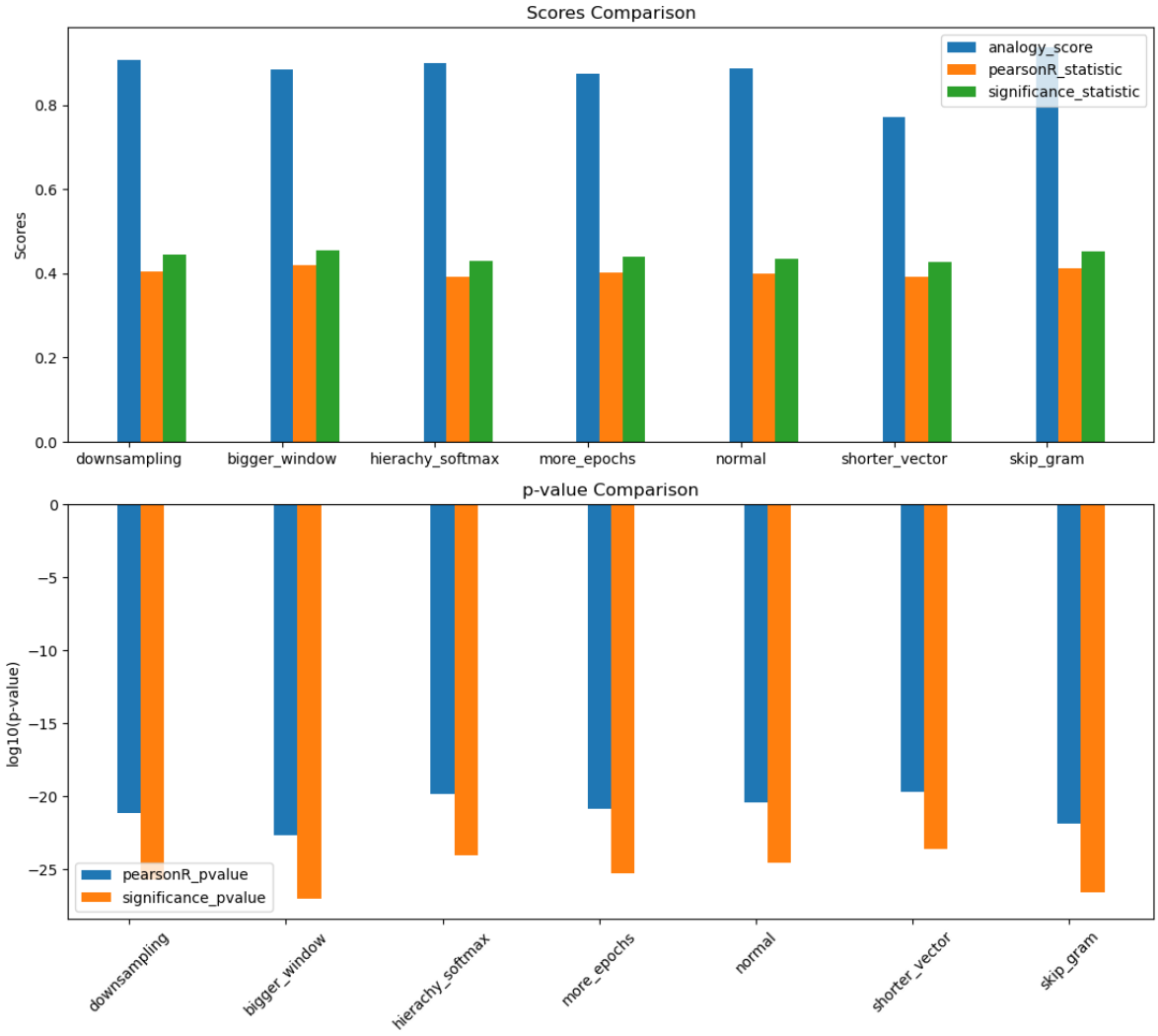
在给出的结果中，有三个项，分别是 `similarity_scores`、`PearsonRResult` 和 `SignificanceResult`，它们的含义如下：

1. `similarity_scores`：这是一个表示词相似度估计结果的值，表示两个词语之间的相似度得分，具体的度量方式可能取决于具体的评估方法和模型。
2. `PearsonRResult`：这是一个包含两个值的元组，用于描述皮尔逊相关系数的结果。皮尔逊相关系数用于衡量两个变量之间的线性相关性。在给定的结果中，它的第一个值是 `statistic`，表示计算得出的皮尔逊相关系数的值，为 0.4192044028992752。第二个值是 `pvalue`，表示计算得出的皮尔逊相关系数的显著性水平，为 2.0282580055611441e-23。
3. `SignificanceResult`：这是一个包含两个值的元组，用于描述显著性检验的结果。在给定的结果中，它的第一个值是 `statistic`，表示计算得出的显著性检验的值，为 0.4549766571598661。第二个值是 `pvalue`，表示计算得出的显著性检验的显著性水平，为 8.850982592087141e-28。

这些值用于描述词相似度估计结果的统计特征和显著性。具体解读这些值的含义需要结合具体的统计方法和领域知识。一般来说，较高的相似度得分和较低的 p 值可以表示两个词语之间的较高相似度和较高的统计显著性。

这里我们用到的词类比测试文件和词相似度测试文件均来自于https://github.com/Leonard-Xu/CWE。

我们将不同模型的得分作图如下：



通过观察评估得分，我们可以得到如下结论：

1. 降采样得到的词向量相对于默认配置的词向量，在词类比估计和词相似度估计上均具有更高的得分。说明通过对缺乏实义的高频虚词进行人为的降采样可以有效提高对词语实义的学习效果，提升最终获得的词向量性能。
2. 增大窗口尺寸的模型相对于默认配置的模型在词类比估计上的得分差别不大，但是在词相似度估计上的得分前者好于后者。说明在一定程度上增大窗口尺寸能够更好地捕捉词语之间的关联性，从而更好地反映词语之间的语义关联信息。
3. 通过增多迭代次数，模型的词类比估计和词相似度估计的得分明显提升。说明在一定程度内进行更多轮的学习能够明显提升结果质量。
4. 短向量模型相对于默认模型在词类比估计和词相似度估计的得分均较低，说明在一定程度上提高向量长度可以丰富其语义表示能力，而短向量可能会更难捕捉更为精确的语义信息。
5. Skip-gram模型相对于默认的CBOW模型在词类比估计和词相似度估计中均具有更高的得分，甚至超过了使用了降采样的CBOW模型。这说明Skip-gram 模型在该任务上的表现要好于CBOW模型。其原因可能为kip-gram 模型通过预测目标词的上下文词来进行训练，因此它能够捕捉到更多的上下文信息。并且Skip-gram 模型对于低频词的处理更为有效。由于低频词在语料库中出现的次数较少，CBOW 模型在训练过程中难以准确地学习到它们的词向量表示。相比之下，Skip-gram 模型通过多次训练样本生成过程中的负采样，能够更好地处理低频词，从而提高了低频词的表示效果。此外，由于 Skip-gram 模型通过上下文预测目标词，它可以在不同的上下文中学习到多义词的不同语义表示。相比之下，CBOW 模型将多义词的不同语义信息平均化，可能导致模糊的表示。

6. Hierarchy Softmax（层次化 Softmax）和负采样（Negative Sampling）两种方法训练出的词向量各方面性能并无显著性的差异，故我们无法由此判断两种训练方法在结果上具有明显的优劣之分。

以上，便是本实验对结果的分析。

Part 5：实验心得

在本实验中，我们深入理解了Word2Vec的训练原理，对CBOW和Skip-gram的训练方法有了理论上的清晰认识。并通过对数据的分析与处理有效将数据清洗得更加适合词向量的训练。并通过自发人为地降采样提高了模型训练的性能。并通过对模型结果的性能分析和对比评估验证了词向量的性质和Word2Vec模型训练的知识，收获颇丰。