

成 绩

****

深度学习与自然语言处理

第四次大作业

|  |  |
| --- | --- |
| 院（系）名称 | 自动化科学与电气工程学院 |
| 专业名称 | 电子信息 |
| 学号 | ZY2103309 |
| 姓名 | 熊方书 |
| 指导教师 | 秦曾昌 |

2022年5月19日

1. **任务描述**

利用给定语料库（或者自选语料库），利用神经语言模型（如：Word2Vec， GloVe等模型）来训练词向量，通过对词向量的聚类或者其他方法来验证词向量的有效性。

1. **实验原理**

计算机无法看懂人类的自然语言，也无法对其进行直接处理，因此自然语言处理第一步需要将自然文字转换为计算机能够“看懂”的数字，因此需要将自然文字进行编码，转换由数字组成的词向量。

1. **Word2Vec工作原理**

2013年来自Google团队的Tomas Mikolov等人设计开发了一款名为Word2vec的程序，该程序可以很简便地设置最终词向量的维度，并根据不同的任务需求设置不同维度数，得到的词向量能满足相加性，可以用公式2-1表示词向量的相加性。



由公式可以看出，词向量之间简单的计算方法就能够表达词向量互相的语义关系。图1简洁形象地展示了词向量的相加性。

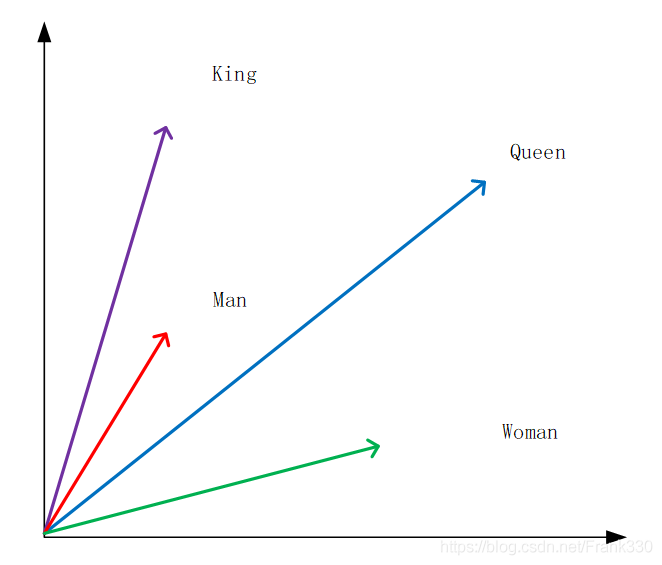
****

图1 词向量的相加性表示

Word2vec是基于统计语言模型而构建的，统计语言模型是用于统计一串字符序列成立的概率大小的概率模型。假定 代表一个字符串，组成成分为 ，记为 ，则由 的联合概率可计算出出现的概率：



利用贝叶斯公式可将上式分解为：



Word2vec是一款利用深度学习来训练生成词向量的工具，它的主要思想是海量的文本中隐含着词与词之间的联系，利用其训练大量文本后可得到质量高且语义相近的词向量。Word2vec的优势是它改进了传统神经网络的词向量语言模型，大大降低了传统向量空间模型处理数据存在的高维度问题。传统的神经网络模型有输入、隐藏和输出三层结构，隐藏层到输出层的softmax层需要计算所有输入词的softmax概率然后判断挑选出最大的值，该过程的计算复杂度很高。相比于传统神经网络，Word2vec在输入层做了改变，传统的神经网络在输入时通常需要先进行线性变换并加入激活函数，而Word2vec则采用求和平均方式处理输入的词向量，将多个词向量转变成一个词向量，大大减小了计算强度；不仅如此，在输出层Word2vec也作了改进，该程序去掉了隐藏层到输出层过程，改用哈夫曼树来取代。假设将哈夫曼树中每个叶子节点赋为权值，则哈夫曼树的目的就是构造出最短的带有权值的路径，哈夫曼又被称为最优二叉树。如图2所示为叶子节点被赋予权值的哈夫曼树。

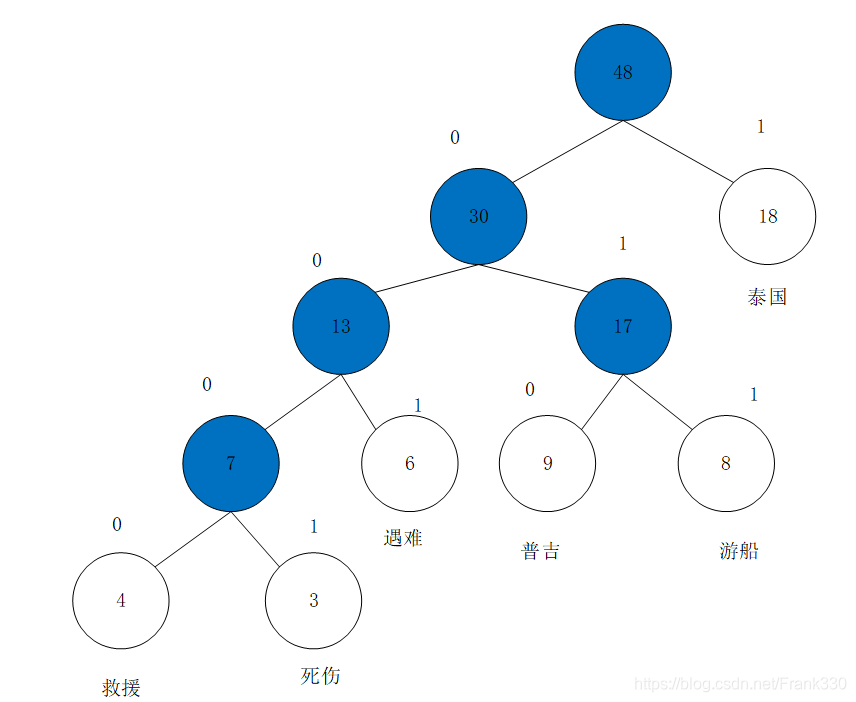


图2 根据权重构造的哈夫曼树

**连续词袋模型（CBOW）**

CBOW的主要思想是在某个语句片段中，在知道某个词上下文的基础上来推测某个特征词出现的概率，也就是说模型的输入是某个词的上下文，而输出是某个词的概率。该模型有输入、投影和输出三层结构。

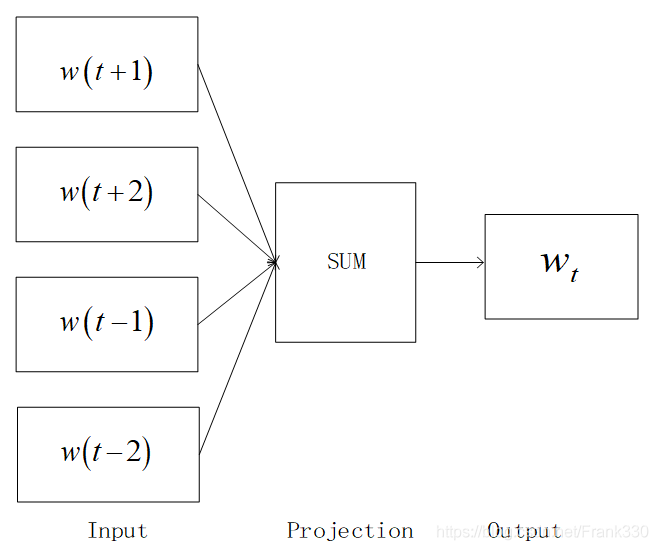
****

图3 CBOW的模型结构图

**Skip-gram模型**

Skip-gram模型也是统计语言模型的一种，处理数据时和CBOW是相反的过程，目的是输入一个特征词，通过模型训练后得到该词的上下文信息。其结构模型图如下所示。

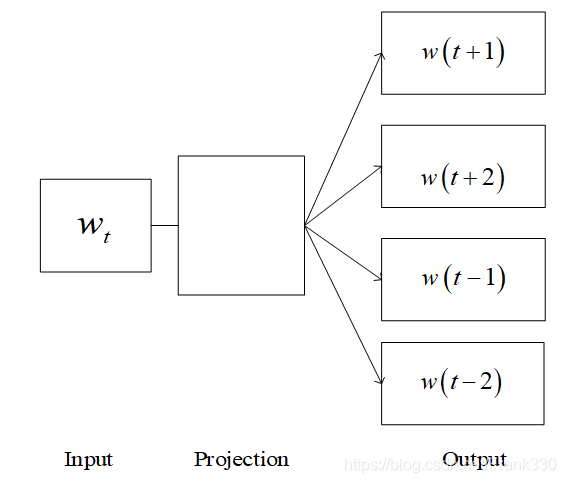


图4 Skip-gram模型结构图

Input层即输入层：输入的数据是语料中的中心词，同CBOW模型一样，也是用单向量表示。

Projection层即投影层：此层在Skip-gram模型中没有起到作用，只是为了在结构上和CBOW模型作对比。

Output层即输出层：此层的结构也是最优二叉树，能更生动地表现词频高的向量。

1. **K-Means聚类**

基本思想：簇内尽量紧凑，簇间尽量分散

算法步骤：1、随机初始化k kk个聚类中心点：可以从样本集S中随机选择，

也可以随机生成。

2、遍历样本集S所有样本点，计算每个样本到中心的距离，并将

该样本分到最近中心点的簇

3、取各个簇内所有样本的均值样本作为新的聚类中心点

4、重复2，3过程，直到达到一定的迭代次数或者聚类中心不再改

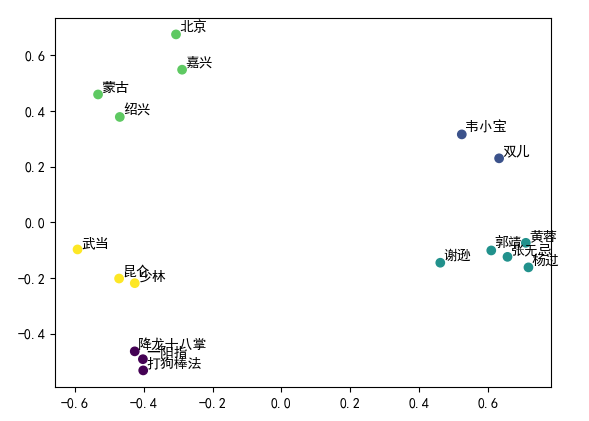
变

1. **实验内容与结果**
2. **实验内容**

​ 本文基于 Python 进行 Word2vec 模型的训练。首先使用jieba对16本金庸小说的合集进行分词，分词处理与前几次作业基本相同，在此不再赘述。分词分句完成后，用gensim中的word2vec进行词嵌入，并打包输出键为单词，值为词向量的字典。得到单词对应的词向量后，对其标准化，再使用 K-Means 对指定单词的词向量进行聚类分析，并将词向量使用 PCA 降维后绘图。

1. **实验结果**

​ 输入keys=['郭靖','黄蓉','杨过','张无忌','谢逊','韦小宝','双儿','少林','武当','昆仑','北京','蒙古','嘉兴','绍兴','降龙十八掌','打狗棒法','一阳指']，选择 K-Means 簇数为 5，得词向量聚类结果如下图：



**附录：代码**

import jieba

import re

from gensim.models import word2vec

import numpy as np

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import pickle as pkl

def cut\_sentences(content):

"""文本分句处理"""

end\_flag = ['?', '!', '.', '？', '！', '。', '…']

content\_len = len(content)

sentences = []

tmp\_char = ''

for idx, char in enumerate(content):

# 拼接字符

tmp\_char += char

# 判断是否已经到了最后一位

if (idx + 1) == content\_len:

sentences.append(tmp\_char)

break

# 判断此字符是否为结束符号

if char in end\_flag:

# 再判断下一个字符是否为结束符号，如果不是结束符号，则切分句子

next\_idx = idx + 1

if not content[next\_idx] in end\_flag:

sentences.append(tmp\_char)

tmp\_char = ''

return sentences

def word\_seg(path, dest):

"""对path指向文本进行基于jieba的分词，返回以空格分隔单词的文本"""

with open(path, "r", encoding="ANSI") as f:

data = f.read()

f.close()

text = cut\_sentences(data)

with open(dest, "w+", encoding="utf-8") as f:

for sentence in text:

sentence = data = re.sub('[^\u4e00-\u9fa5]+', '', sentence)

f.write(" ".join(jieba.lcut(sentence, use\_paddle=True, cut\_all=False)) + '\n')

def vec\_gen(path):

"""根据分句分词完成的文件路径生成词向量字典"""

train\_data = word2vec.LineSentence(path)

model = word2vec.Word2Vec(train\_data,

vector\_size=100,

window=5,

workers=4)

model.wv.vectors = model.wv.vectors / (np.linalg.norm(model.wv.vectors, axis=1).reshape(-1, 1))

vec\_dist = dict(zip(model.wv.index\_to\_key,model.wv.vectors))

with open('VEC/vec\_dist', 'wb') as f:

pkl.dump(vec\_dist,f)

def cluster(keys, n\_clusters):

with open('VEC/vec\_dist', 'rb') as f:

vec\_dist = pkl.load(f)

vec = []

for k in keys:

vec.append(vec\_dist[k])

label = KMeans(n\_clusters=n\_clusters).fit\_predict(vec)

vec = PCA(n\_components=2).fit\_transform(vec)

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

plt.scatter(vec[:,0],vec[:, 1],c=label)

for i, w in enumerate(keys):

plt.annotate(s=w, xy=(vec[:, 0][i], vec[:, 1][i]),

xytext=(vec[:, 0][i] + 0.01, vec[:, 1][i] + 0.01))

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# s = word\_seg('VEC/text/merged.txt','VEC/seg.txt')

# wv = vec\_gen('VEC/seg.txt')

cluster(['郭靖','黄蓉','杨过','小龙女','郭襄','张无忌','谢逊','韦小宝','双儿','康熙','少林','武当','昆仑','北京','嘉兴','杭州','扬州','降龙十八掌','打狗棒法','一阳指'], 5)