

介绍：（to be cont.）

整个模型是从UN Economic and Social Council (ECOSOC)的文章中寻找SDG相关的词汇，来寻找SDG之间的关系。为了实现这样的模型，我们采用的词向量模型是Word2Vec模型。Word2Vec 是一种流行的自然语言处理技术，它可以将自然语言中的单词映射到低维向量空间中，使得单词在向量空间中的相对位置可以表示它们在语义上的相似性和相关性。除此之外Word2Vec使用了一种简单但高效的算法，称为Skip-gram或Continuous Bag-of-Words（CBOW）。这些算法不需要大量计算，因此Word2Vec的训练速度比其他模型更快。且与其他词向量模型相比，Word2Vec具有更好的性能。因此，Word2Vec 可以应用于我们的SDGs分析场景。但在实际使用中绝大部分项目会选择CBOW. 但由于我们训练用的语料库较小且关系更为复杂，所以我们选择skip-gram。因为在skip-gram中，每个单词都有一个独立的向量表示，因此每个向量可以专门捕捉该单词在语料库中的不同语义，包括它在不同上下文中的用法。

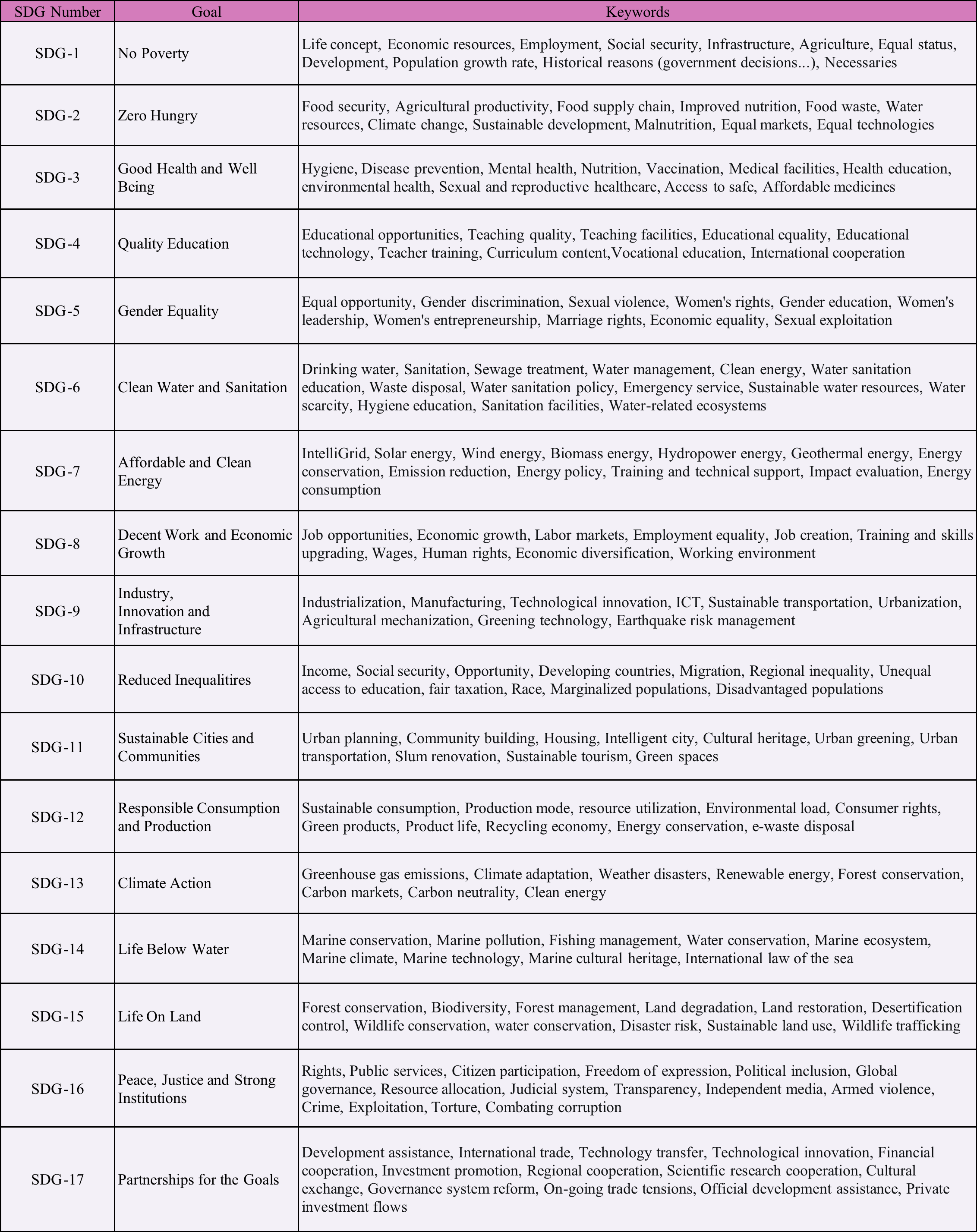
数据来源：

我们从ECOSOC发布的文章中选取了最近发布的57篇文章使用Natural Language Processing(NLP)技术进行分析。我们选取ECOSOC的文章的原因是其提供了广泛的数据，涵盖了各种可持续发展目标及其相关指标。这些数据不仅包括国际层面的数据，也包括国家层面的数据，因此可以对不同地区和国家的进展情况进行比较分析。并且，ECOSOC提供的数据基于标准化的指标和测量方法，可以确保数据在不同地区和国家之间具有一致性和可比性，从而更容易进行分析和评估。除此之外，ECOSOC提供的数据是公开可用的，数据源可以追溯到其来源，使得研究人员和政策制定者可以验证数据的准确性和可靠性。因此，我们选择根据ECOSOC近期发布的57篇文章的关键词来寻找17个SDG之间的network of the relationship.

详细步骤：

1. 目标定位

我们根据联合国的17个SDG，将每个SDG细分为若干个与其相关的关键词，在我们选择57篇文章当中筛选相关关键词的出现频率。



1. 文章预处理

\*加公式 (laf

我们认为分词算法根据其核心思想主要分为两种，第一种是基于字典的分词，先把句子按照字典切分成词，再寻找词的最佳组合方式；第二种是基于字的分词，即由字构词，先把句子分成一个个字，再将字组合成词，寻找最优的切分策略，同时也可以转化成序列标注问题。归根结底，上述两种方法都可以归结为在图或者概率图上寻找最短路径的问题。这里，针对SDGs，因为我们都是从联合国官方文档里面获取相应的信息，所以，第一种方法可能更加适合我们的情况。最终，我们选择了这种情况。

文章预处理是指对原始文本进行处理，以便更好地应用自然语言处理（NLP）技术。文章预处理的目的是为了提高NLP任务的准确性和效率，减少错误和噪声，并为后续的数据分析和挖掘提供更加清晰的数据。将文章转换为机器可读的格式。对此，我们使用Python中的Natural Language Toolkit (NLTK)库对文章根据SDGs划分的关键词进行预处理，包括去除标点符号，分词，去除停用词等。此外，将文本分解成单词或标记，以便更好地理解文本的语义和结构。分词也可以帮助我们更好地理解句子和文档的含义。

同时，我们还需要进行词形还原和词干提取：在不同的形式下，同一个单词可能出现多次，如“running”和“runs”，它们的原形都是“run”。这样对数据进行处理因此，为了提高准确性，需要对单词进行词形还原或词干提取，使它们归一化为相同的形式。

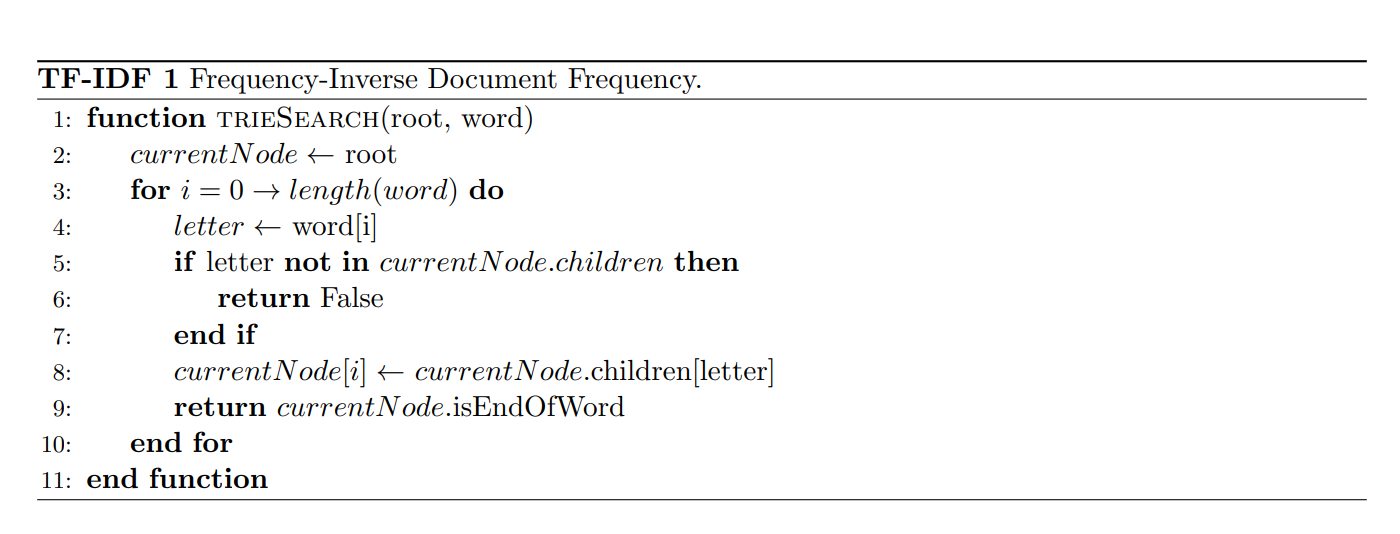
此外，这些我们从联合国官网获取的文本数据通常包含一些不必要的信息，如标点符号、HTML标记、表情符号、网址等。这些噪声会干扰自然语言处理算法的准确性，因此需要去除。将句子分割成单词或子词，以便计算机可以理解和处理。在英文中，通常是以空格或标点符号为分隔符。这里，我们采用jieba词库进行分词，jieba采用了基于Trie树的分词算法，具有较高的分词速度和较小的内存占用。Trie树是基于以下搜索字典项目：

1. 从根节点开始第一次搜索
2. 取得要查找关键词的第一个字母，并根据该字母选择对应的子树并转到该子树继续进行检索
3. 在相应的子树上，取得要查找关键词的第二个字母,并进一步选择对应的子树进行检索。
4. 选代过程......
5. 在某个结点处，关键词的所有字母已被取出，则读取附在该结点上的信息，即完成查找其他操作类似处理

接着，我们还会采用停用词过滤的方法，停用词是指在文本中出现频率非常高，但并没有实际意义的词语，如“的”、“是”、“在”等。过滤掉这些停用词可以降低数据的噪声和冗余，提高后续处理的效率和质量。摆脱停用词对于任何自然语言处理任务都非常有意义。NLTK还允许用户根据自己的需要定制停止词列表。例如，他们可以从列表中添加或删除单词，或者根据自己的文本语料库创建一个新的列表。因此，我们会选择NLTK作为停用词表。

词干提取和词形还原：将单词的不同形态（如时态、复数形式）转换成其原始形式，以便提高文本的准确性和一致性。词干提取是指将单词缩减到其基本形式，如将“running”和“runs”都转换为“run”，而词形还原则是指将单词恢复到其原始形式，如将“am”, “is”, “are”都还原为“be”。

here's the pseudocode for searching an item in a Trie tree dictionary:

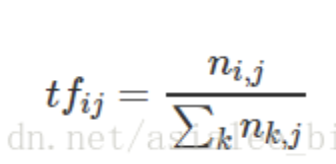


In this pseudocode, root is the root node of the Trie tree and word is the item that we want to search for in the dictionary. The function trieSearch iterates through each letter of the word and traverses the Trie tree accordingly. If a letter is not found in the currentNode's children, then the search terminates and the function returns False. If all letters in the word are found and the final node is marked as the end of a word, then the function returns True.

1. 筛选关键词

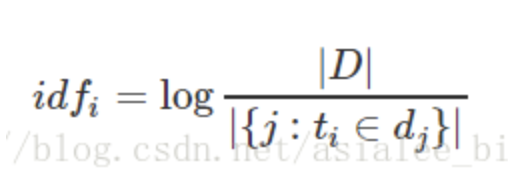
为了更好的寻找到文章的关键词，需要的文章进行处理。我们采用Term Frequency-Inverse Document Frequency(TF-IDF)算法，便于区分字词的重要性, 以此来筛选具有良好的类别区分能力词。TF-IDF算法能够准确地反映一个词汇在文档中的重要性。相比于仅使用Term Frequency，TF-IDF算法结合了Term Frequency和Inverse Document Frequency两个因素，可以更好地反映一个词汇在文档集合中的重要性。例如，对于一些高频词汇（如“the”、“be”等），它们在文档中出现的次数很多，但是它们在整个文档集合中出现的次数也很多，因此它们的TF-IDF值往往较低，可以被有效地过滤掉。下面是算法的具体步骤：

第一步，需要统计词汇出现的数量，也就是计算Term Frequency。在统计到文章词汇数量后，由于文章长短不齐，为了防止文章长短对数据造成一定影响，需要对该数据进行normalized processing。公式如下：



**N（i,j）**是该词在文章中出现的次数，分母则是文章中所有词汇出现的次数总和

第二步，我们需要计算Inverse Document Frequency（IDF），由**总文件数目除以包含该词语的文件的数目**，**再将得到的商取对数得到**。词汇的IDF越大，说明其类别区分能力越好。IDF计算公式如下：



**ps:(这里分母应该是1+|{j:ti∈dj}|**

**|D|** **是文章总数**。**|{j:ti∈dj}| 表示包含该词汇的文章数目**

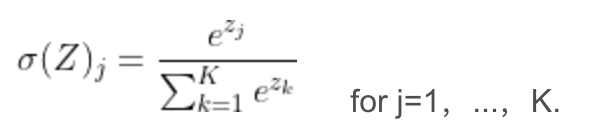
将TF与IDF相乘即可过滤掉常见的词汇。由此，我们可以提取每篇文章的关键词。

伪代码：

|  |
| --- |
| # 输入参数:  # - corpus: 文档集合  # - stopwords: 停用词列表  # 计算每个文档中每个词的词频  tf = {}  for doc in corpus:  tf[doc] = {}  for word in doc.split():  if word not in stopwords:  tf[doc][word] = tf[doc].get(word, 0) + 1  # 计算每个词在文档集合中的逆文档频率  idf = {}  for doc in corpus:  for word in tf[doc]:  if word not in idf:  idf[word] = 1  else:  idf[word] += 1  for word in idf:  idf[word] = log(len(corpus) / idf[word])  # 计算每个文档中每个词的TF-IDF值  tfidf = {}  for doc in corpus:  tfidf[doc] = {}  for word in tf[doc]:  tfidf[doc][word] = tf[doc][word] \* idf[word] |

1. 训练Word2Vec模型

首先使用NLTK库构建词汇表，将每个单词映射到唯一的整数标识。接下来就是训练Word2Vec模型，设置skip-window=4，Hidden Layer=90。为了评估可持续发展目标联合国报告中的重叠与科学文献中可持续发展目标之间的整合之间的一致性，我们收集了Google Scholars及ECOSOC发布的其他文章，以此作为训练数据。我们使用TensorFlow作为工具来训练Word2Vec skip-gram模型。这里需要将上述所有文章作为输入。对外，我们采用one-hot进行编码将所有的字符转化成二进制的编码形式。再将one-hot编码后的词向量，通过神经网络的hidden layer，映射到一个低纬度的空间。再将hidden layer和输出层进行全连接。然后，我们需要将向量转换为概率分布的函数softmax用于计算当前单词与其他单词的相似度，以确定最可能的上下文单词。softmax函数可以表示为：



z就是我们的相似度

w\_c表示目标单词的one-hot向量

v\_c表示目标单词的词向量

u\_x表示除目标单词外第x个单词的词向量

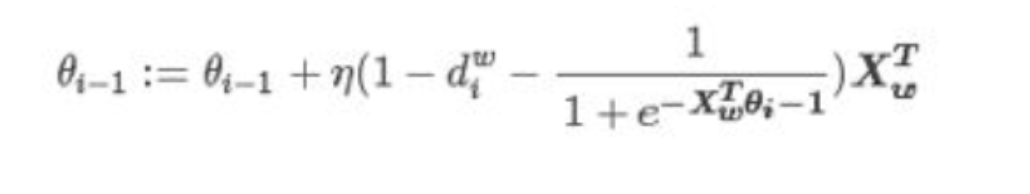
W表示目标单词矩阵

W'表示其他单词矩阵

词向量的维度是d

词汇表的维度是V

接下来，利用梯度上升法求似然函数的最大值。最后，我们需要更新Skip-gram模型的参数。更新参数的公式如下：

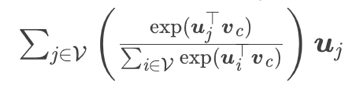


𝞋模型参数，一开始随机初始化

X(w)词向量

更新中心词的向量如下：（把这两个合并起来





V(c 中心词向量

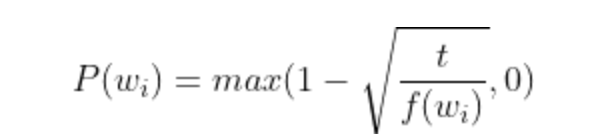
*uo*​背景词矩阵

背景词的索引为o

j的取值范围是− m ≤ j ≤ m , j不能为0

当梯度一旦收敛，停止更新，此时概率值为极大值，词向量为最优解。

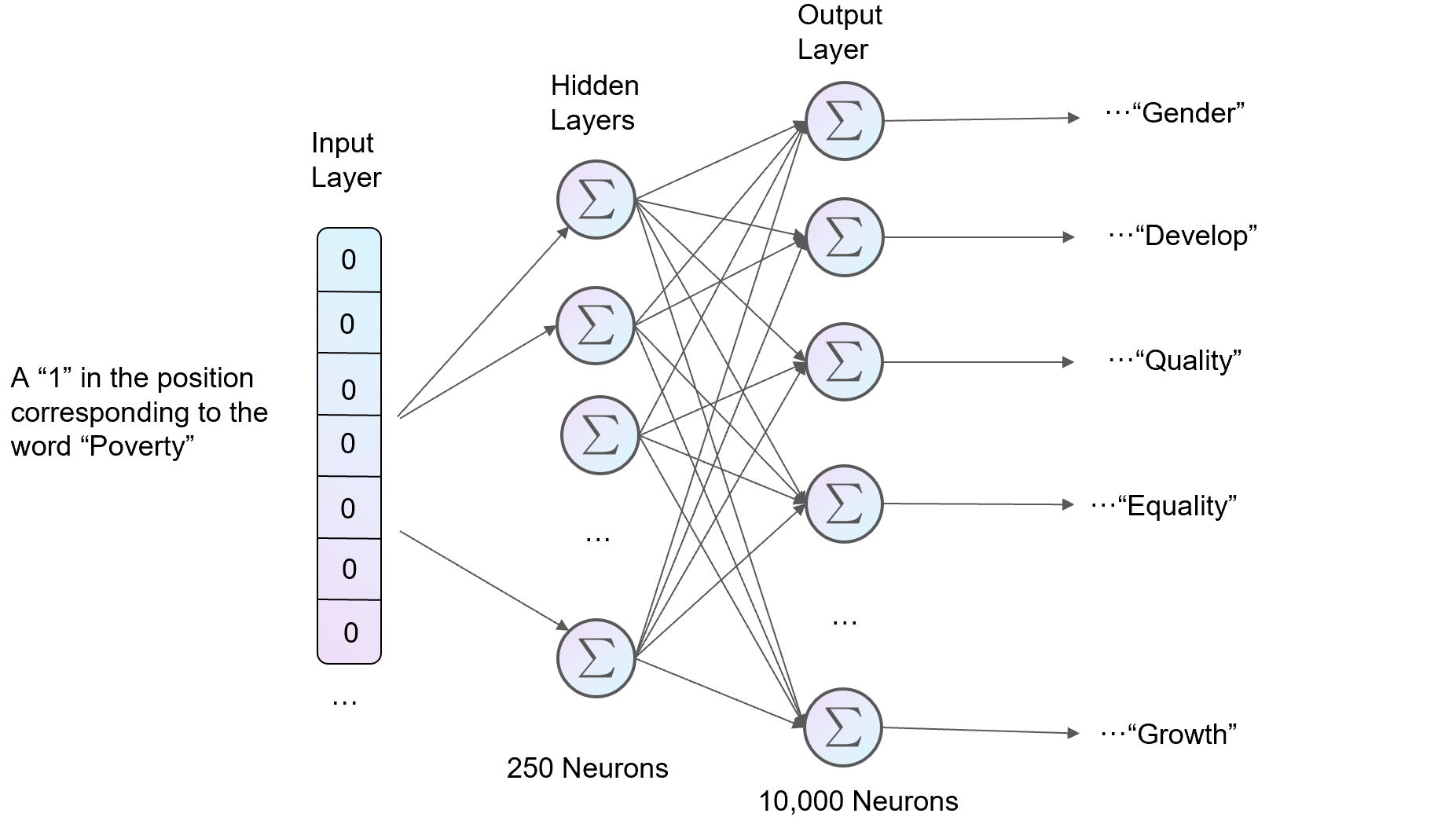
除此之外，我们还将使用二次采样法处理原始文本，这也是为了降低高频词在语料中出现的频次。方法是随机将高频的词抛弃，频率越高，被抛弃的概率就越大；频率越低，被抛弃的概率就越小。标点符号或冠词这样的高频词就会被抛弃，从而优化整个词表的词向量训练效果。具体来说，数据集中每个被索引词 w\_i将有一定概率被丢弃，该丢弃概率为：



其中f(w\_i)是w\_i 的个数与总词数之比

常数 t 是一个参数（此处设置为10^-4）。

So, the process actually looks like this:



搬一个word2vec伪代码 (tqq

（简化版

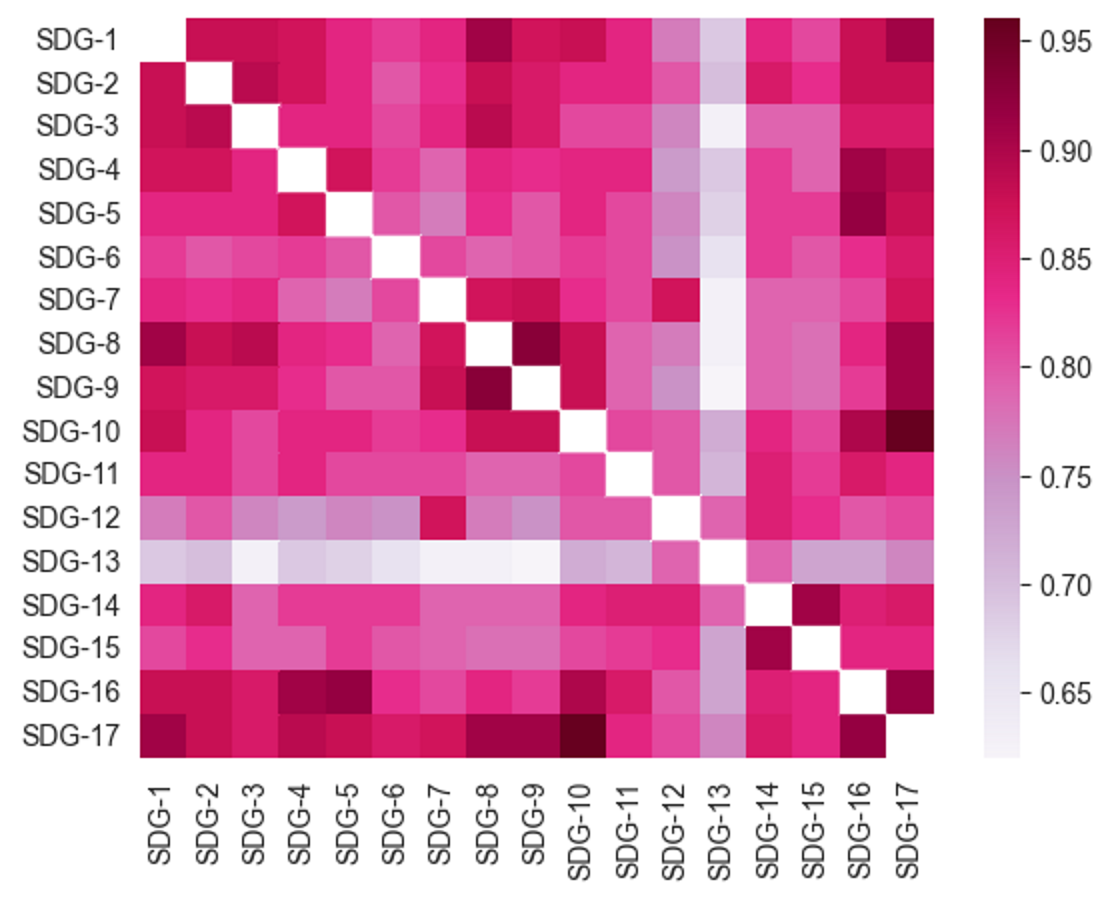
|  |
| --- |
| # 构建词汇表和单词映射  vocab = set([word for sentence in corpus for word in sentence.split()])  word2idx = {}  idx2word = {}  for i, word in enumerate(vocab):  word2idx[word] = i  idx2word[i] = word  # 初始化词向量和上下文向量  embedding\_size = 100  vocab\_size = len(vocab)  W1 = np.random.uniform(-0.5/embedding\_size, 0.5/embedding\_size, (vocab\_size, embedding\_size))  W2 = np.random.uniform(-0.5/embedding\_size, 0.5/embedding\_size, (embedding\_size, vocab\_size))  # 定义训练数据  window\_size = 2  data = []  for sentence in corpus:  tokens = sentence.split()  for i, center\_word in enumerate(tokens):  for j in range(max(0, i - window\_size), min(i + window\_size + 1, len(tokens))):  if j != i:  context\_word = tokens[j]  data.append((center\_word, context\_word))  # 训练模型  learning\_rate = 0.01  epochs = 5  for epoch in range(epochs):  np.random.shuffle(data)  for center\_word, context\_word in data:  center\_word\_idx = word2idx[center\_word]  context\_word\_idx = word2idx[context\_word]  # 前向传播  center\_word\_vec = W1[center\_word\_idx]  context\_word\_vec = W2[:, context\_word\_idx]  pred = np.dot(center\_word\_vec, context\_word\_vec)  pred\_prob = sigmoid(pred)  # 反向传播  grad = learning\_rate \* (pred\_prob - 1)  grad\_w2 = np.outer(center\_word\_vec, grad)  grad\_w1 = np.dot(W2[:, context\_word\_idx], grad)  W2[:, context\_word\_idx] -= grad\_w2  W1[center\_word\_idx] -= grad\_w1  # 定义sigmoid函数  def sigmoid(x):  return 1 / (1 + np.exp(-x)) |

模型实现：

在训练的过程中，我们以25位单位尝试了从50到300个隐藏层数量当中的最好的，迭代的增加了模型的隐藏层数量 which makes the model neither 过拟合 nor 欠拟合。直到我们达到嵌入产生稳定聚类的点才停止训练。

Finally, We set up a Doc2vec model with 90 hidden layers. Considering the relatively small number of documents, we also performed a sensitivity analysis when training the doc2vec model to ensure the stability of the results.

归一化欧几里得距离 (laf



模型评价（优势/劣势

敏感性分析 （不太懂，慢慢查

附录:

