EvoLDA-DB model

import numpy as np

import glob

import os

import jieba

import jieba.posseg as pseg # 引入词性标注模块

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

import time

# 定义一些超参数

K = 25 # 主题数目

alpha = 0.2 # 文档主题分布的超参数

beta = 0.01 # 主题词分布的超参数

num\_iters = 1000 # 迭代次数

data\_path = "/Users/Desktop/postgraduate/D-NewNlp/datas/Document\_No. 1\_of\_the\_Central\_Committee\_of\_China/1"

# 读取所有文本文件

documents = []

total\_word\_count = 0

# 添加一个空字典，用于存储每个单词出现的次数

word\_freq = {}

for filename in os.listdir(data\_path):

if filename.endswith(".txt"):

with open(os.path.join(data\_path, filename), "r") as f:

text = f.read()

# 分词并进行词性标注

words = []

for word, pos in pseg.cut(text):

# 仅保留名词

if pos.startswith('n'):

if word not in word\_freq:

word\_freq[word] = 1

else:

word\_freq[word] += 1

p = word\_freq[word] / sum(word\_freq.values())

if 0.001 < p < 0.999:

words.append(word)

doc\_str = " ".join(words)

if len(doc\_str) > 0:

documents.append(doc\_str)

total\_word\_count += len(words)

print("The number of words in doc\_str is:", total\_word\_count)

# 构建词表并特征提取

vectorizer = CountVectorizer()

word\_freq = vectorizer.fit\_transform(documents)

vocab = vectorizer.get\_feature\_names\_out()

vocab\_size = len(vocab)

word2idx = {w: i for i, w in enumerate(vocab)}

tf\_transformer = TfidfTransformer(use\_idf=False)

word\_freq\_tf = tf\_transformer.fit\_transform(word\_freq)

data = np.asarray(word\_freq\_tf.toarray(), dtype=np.float32)

# 初始化文档-主题和主题-词语分布

doc\_topic\_dist = np.random.dirichlet([alpha] \* K, len(documents))

topic\_word\_dist = np.random.dirichlet([beta] \* vocab\_size, K)

i = 0

# 迭代推断

for \_ in range(num\_iters):

start\_time = time.time()

# E 步骤：计算每个词语属于每个主题的概率

topic\_word\_prob = np.zeros((K, vocab\_size))

for k in range(K):

for w in range(vocab\_size):

try:

topic\_word\_prob[k, w] = np.sum([doc\_topic\_dist[d, k] \* topic\_word\_dist[k, w] for d, doc in enumerate(documents) if vocab[w] in doc])

except KeyError:

continue

topic\_word\_prob[k] /= topic\_word\_prob[k].sum()

# M 步骤：更新主题-词语分布和文档-主题分布

for k in range(K):

for w in range(vocab\_size):

topic\_word\_dist[k, w] = (topic\_word\_prob[k, w] \* np.array([doc\_topic\_dist[d, k] for d in range(len(documents))])).sum()

topic\_word\_dist[k] /= topic\_word\_dist[k].sum()

for d in range(len(documents)):

for k in range(K):

doc\_topic\_dist[d, k] = np.sum((topic\_word\_prob[k, [word2idx[word] for word in documents[d].split() if word in word2idx]][:, np.newaxis] \* topic\_word\_dist[k]))

doc\_topic\_dist[d] /= doc\_topic\_dist[d].sum()

end\_time = time.time()

iter\_time = end\_time - start\_time

i = i+1

print("Iteration {} time: {:.2f} seconds".format(i, iter\_time))

# 选取每个主题的主题词

topic\_words = []

for k in range(K):

topic\_word\_prob = topic\_word\_dist[k]

sorted\_prob\_indices = np.argsort(topic\_word\_prob)[::-1]

topic\_word\_list = [vocab[i] for i in sorted\_prob\_indices] # 选取所有词

topic\_words.append(topic\_word\_list)

# 输出每个主题的主题词

for k in range(K):

print("Topic {}: {}".format(k, ", ".join(topic\_words[k])))

import pandas as pd

# 计算所有词的词频并存储在一个字典中

word\_freq\_dict = {}

for doc\_str in documents:

for word in doc\_str.split():

if word in word\_freq\_dict:

word\_freq\_dict[word] += 1

else:

word\_freq\_dict[word] = 1

# 创建 Excel 文件并写入数据

writer = pd.ExcelWriter('/Users/wuyehui/Desktop/postgraduate/D-NewNlp/datas/Document\_No. 1\_of\_the\_Central\_Committee\_of\_China/1/topic\_words1.xlsx', engine='xlsxwriter')

for k in range(K):

top\_words\_idx = np.argsort(-topic\_word\_dist[k])

top\_words = [vocab[i] for i in top\_words\_idx]

# 计算每个单词的概率并将其存储在字典中

word\_prob\_dict = {}

prob\_sum = 0

for word in top\_words:

if word in word\_freq\_dict:

prob = word\_freq\_dict[word] / total\_word\_count

prob\_sum += prob

word\_prob\_dict[word] = prob

prob\_sum\_str = f'P={prob\_sum:.5f}'

# 创建新的工作簿并写入数据

sheet\_name = f'Topic {k}'

df = pd.DataFrame({'Word': top\_words, 'Probability': [word\_prob\_dict.get(word, '') for word in top\_words]})

df.to\_excel(writer, sheet\_name=sheet\_name, index=False)

worksheet = writer.sheets[sheet\_name]

worksheet.write(len(top\_words)+1, 0, prob\_sum\_str)

# 保存 Excel 文件并关闭 writer

writer.save()

from gensim.models import Word2Vec, KeyedVectors

# 读取预训练的词向量文件，每行为一个词语及其对应的向量

embedding\_file\_path = "/Users/wuyehui/Downloads/sgns.renmin.bigram"

word\_vectors = KeyedVectors.load\_word2vec\_format(embedding\_file\_path, binary=False, encoding='utf-8', unicode\_errors='ignore')

# 创建一个空的Word2Vec模型

model = Word2Vec(vector\_size=300, window=5, min\_count=1, workers=4)

# 填充模型的词汇表和词向量

model.wv.add\_vectors(keys=word\_vectors.index\_to\_key, weights=word\_vectors.vectors)

# 保存模型为二进制文件

model.save("/Users/wuyehui/Downloads/word2vec.model")

import pandas as pd

from gensim.models import Word2Vec

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from gensim.models import Word2Vec

from gensim.models.keyedvectors import KeyedVectors

import pandas as pd

import numpy as np

from gensim.models import Word2Vec

from sklearn.decomposition import PCA

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

# 读取Excel文件，假设第一列是您的语料库

df = pd.read\_excel("/Users/wuyehui/Desktop/D-LDA语料库合并/合并（无字）.xlsx", header=None, names=["words"])

# 加载中文词向量文件（Numpy数组）

word\_vectors = np.load("/Users/wuyehui/Downloads/word2vec.model.wv.vectors.npy")

# 创建词向量索引

word2idx = {word: idx for idx, word in enumerate(df["words"])}

# 将词语转换为词向量

def get\_word\_vector(word):

try:

index = word2idx[word] # 获取词语在词向量文件中的索引

return word\_vectors[index]

except KeyError:

return None

df["vector"] = df["words"].apply(get\_word\_vector)

# 去除没有对应词向量的词语

df = df.dropna()

# 使用DBSCAN进行密度聚类

eps = 0.5 # 指定邻域半径，可以根据需要调整

min\_samples = 7 # 指定最小样本数，可以根据需要调整

dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)

df["cluster"] = dbscan.fit\_predict(df["vector"].tolist())

# 使用PCA进行降维，以便可视化

pca = PCA(n\_components=2)

reduced\_features = pca.fit\_transform(df["vector"].tolist())

df[["pca1", "pca2"]] = reduced\_features

# 将聚类结果输出到新的Excel文件

df.to\_excel("/Users/wuyehui/Desktop/postgraduate/D-NewNlp/datas/Document\_No. 1\_of\_the\_Central\_Committee\_of\_China/DBSCAN.xlsx", index=False)

# 绘制散点图

plt.scatter(df["pca1"], df["pca2"], c=df["cluster"], cmap="viridis")

plt.xlabel("PCA1")

plt.ylabel("PCA2")

plt.title("DBSCAN Clustering")

plt.show()