#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Dec 4 09:19:40 2019

@author: hym0509

"""

一 命令

1, data

1.1 stopwords

1.2 term frequency(TF)-inverse document frequency(IDF)=TF\*IDF

TF= 某词在文章中出现的次数/文章的总词数

IDF=log(语料库的文本总数/包含该词的文本数+1)

1.3 similarility

(1)sentance A:i like watching tv, but hate watching movie.

sentance B:i hate watching tv, and hate watching movie.

(2)分词：

sentance A:i /like /watching /tv, but/ hate/ watching/ movie.

sentance B:i /hate /watching /tv, and/ hate /watching/ movie.

（3）语料库：[i,like,watching,tv,movie,but,and,hate]

(4) term frequency

sentance A:i(1) like(2) watching(2) tv(1) movie(1) but(1) and(0) hate(1) .

sentance B:i(1) like(0) watching(2) tv(1) movie(1) but(0) and(1) hate(2) .

(5)word frequency vector

sentance A:[1,2,2,1,1,1,0,1]=a

sentance B:[1,0,2,1,1,0,1,2]=b

(6)document term matrix(DTM)=[a b]

2 write,read,csv

2.1 write

2.1.1

f=open('text.txt','w')

f.write('text''\n''mining''\n\'with''\n')

f.close()

2.1.2 追加内容

f=open('text.txt','a')

f.write('python''\n')

f.close()

2,1.3 cover 原内容

f=open('text.txt','w')

f.write('1''\n''2''\n\'3''\n')

f.close()

2.2 read

2.2.1 file.read()

f=open('text.txt')

f.read()

2.2.2 file.readline()

f=open('text.txt')

f.readline()

2.2.3 file.readlines()

f=open('text.txt')

f.readlines()

2.2.4 list()

f=open('text.txt')

l=list(f)

print(f)

2.2.5 不用close的保存方法

with open('text.txt','r') as f:

data=f.readlines()

print(data)

3 jieba function

pip install jieba

3.1 jieba.cut

import jieba

seg\_list = jieba.cut("我爱文本数据分析")

seg\_list

for i in jieba.cut("我爱文本数据分析"):

print (i)

for i in jieba.cut("我爱文本数据分析",cut\_all=True): # 全模式

print (i)

3.2 jieba.cut\_for\_search

seg\_list = jieba.cut\_for\_search("我爱文本数据分析")

seg\_list

for i in jieba.cut\_for\_search("我爱文本数据分析"):

print (i)

3.3 jieba.lcut("sentence")

jieba.lcut("我爱文本数据分析")

print ("/".join(jieba.lcut("我爱文本数据分析")))

3.4 jieba.lcut\_for\_search("sentence")

jieba.lcut\_for\_search("我爱文本数据分析")

3.5 jieba.tokenize(u"sentence")

jieba.tokenize(u"我爱文本数据分析")

for i in jieba.tokenize(u"我爱文本数据分析"):

print (i)

3.6 jieba.suggest\_freq("word")

print ("/".join(jieba.cut("结巴分词是很好的中文分词工具")))

jieba.suggest\_freq(("结巴分词"), True)

print ("/".join(jieba.cut("结巴分词是很好的中文分词工具")))

4 posseg package

4.1

import jieba

from jieba import posseg

pos = list(jieba.posseg.cut("我爱文本数据分析"))

pos

jieba.posseg.lcut("我爱文本数据分析")

4.2 observe dataset

import pandas as pd

train\_df = pd.read\_csv('sohu\_train.txt', sep='\t', header=None)

train\_df.head()

4.3 observe data

for name, group in train\_df.groupby(0):

print(name,'\t', len(group))

5 gensim库构建DTM

pip install gensim

5.1 build dictionary

import gensim

from gensim import corpora

corpus =['This is a text mining book',

'Is this a text mining book',

'Text mining with sklearn']

texts = [[word for word in document.lower().split()] for document in corpus]

dictionary = corpora.Dictionary(texts)

5.2 build DTM

word\_count = [dictionary.doc2bow(text) for text in texts]

word\_count

5.3 matrix transpose

from gensim.matutils import corpus2dense

corpus\_matrix=corpus2dense(word\_count, len(dictionary))

corpus\_matrix.T

6 gensim库构建TF-IDF

from gensim import corpora,models

documents = ['This is a text mining book',

'Is this a text mining book',

'Text mining with python']

texts = [[word for word in document.lower().split()] for document in documents]

dictionary = corpora.Dictionary(texts)

corpus = [dictionary.doc2bow(text) for text in texts]

tfidf\_model = models.TfidfModel(corpus)

corpus\_tfidf = tfidf\_model[corpus]

from gensim.matutils import corpus2dense

corpus\_matrix=corpus2dense(corpus\_tfidf, len(dictionary))

corpus\_matrix.T

7 classifier machine learning(scikit-learn)

7.1 naive bayes

7.1.1 gaussian naive bayes

#以X，Y为trainning set去训练分类模型

import sklearn

from sklearn.naive\_bayes import GaussianNB

import numpy as np

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

Y = np.array([1, 1, 1, 2, 2, 2])

clf = GaussianNB()

clf.fit(X, Y)

#预测testing set的分类

clf.predict(np.array([[-5,-0.1],[5,0.1]]))

#预测testing set被分到各类的概率

clf.predict\_proba( np.array([[-5,-0.1],[5,0.1]]))

#testing set的特征值

X\_test= np.array([[-5,-0.1],[5,0.1]]) # 测试集特征值

Y\_test=np.array([1, 2]) # 测试集真实分类

clf.score(X\_test,Y\_test) # 计算平均准确率

7.1.2 multinomial naive bayes

from sklearn.naive\_bayes import MultinomialNB

X = np.random.randint(5, size=(6, 100))

y = np.array([1, 2, 3, 4, 5, 6])

clf = MultinomialNB()

clf.fit(X, Y)

X\_test=np.random.randint(5, size=(2, 100))

clf.predict(X\_test)

7.1.3 bernoulli naive bayes

from sklearn.naive\_bayes import BernoulliNB

X = np.random.randint(2, size=(5, 20)) # 随机生成二值型训练集

Y = np.array([1, 2, 3, 4, 4])

clf = BernoulliNB()

clf.fit(X, Y)

X\_test=np.random.randint(2, size=(2, 20))

clf.predict(X\_test)

7.2 k-neighbors classifier

from sklearn.neighbors import KNeighborsClassifier

X = np.array([[0,0], [1,0.5], [2,4], [3,3.7]])

y = np.array([0, 0, 1, 1])

clf\_knn = KNeighborsClassifier(n\_neighbors=2)

clf\_knn.fit(X,y)

clf\_knn.predict([[0.5,0.5],[5,5]])

clf\_knn.predict\_proba([[0,3],[1,5]])

7.3 support vector machine

from sklearn.svm import SVC

X = np.array([[-1, -1, -1], [-2, -1, -2], [1, 1, 1], [2, 1, 2]])

y = np.array([1, 1, 2, 2])

clf\_svc = SVC()

clf\_svc.fit(X, y)

clf\_svc.support\_vectors\_

clf\_svc.decision\_function(X)

X\_test = np.array([[-3, 0, -1], [5, 4, 3]])

clf\_svc.predict(X\_test)

7.3 decision tree

from sklearn import tree

X = [[0, 0, -1], [1, 0, 1]]

Y = [0, 1]

clf\_DecisionTree = tree.DecisionTreeClassifier()

clf\_DecisionTree.fit(X, Y)

clf\_DecisionTree.predict([[2, 2, 0]])

8 cluster

8.1 k-means

from sklearn.cluster import KMeans

import numpy as np

X = np.array([[1, 2, 3], [1, 3, 4], [1, 2, 0],

[4, 5, 2], [4, 4, 6], [4, 5, 0]])

kmeans = KMeans(n\_clusters=2)

kmeans.fit(X)

kmeans.fit\_predict(X)

kmeans.cluster\_centers\_

kmeans.inertia\_

8.2 birch

from sklearn.cluster import Birch

X = np.array([[0, 0, 1], [0.5, 0.3, 1], [-0.2, -1, 1], [0, -0.4, -1], [0.3, 0.2, -1]])

birch = Birch(n\_clusters=None)

birch.fit(X)

birch.predict(X)

9 cluster效果评价

9.1 supervised

from sklearn import metrics

labels\_true = [0, 0, 1, 1, 1]

labels\_pred = [0, 0, 1, 1, 2]

9.1.1 adjusted rand index

metrics.adjusted\_rand\_score(labels\_true, labels\_pred)

9.1.2 mutual information

metrics.normalized\_mutual\_info\_score(labels\_true, labels\_pred)

9.1.3 homogeneity

metrics.homogeneity\_score(labels\_true, labels\_pred)

9.1.4 completeness

metrics.completeness\_score(labels\_true, labels\_pred)

9.2 unsuperized

import numpy as np

x=np.array(([5,3,1,0],[4,3,1,1],[-4,3,1,0],[-4,3,1,2],[-5,3,1,2]))

y=np.array([0,0,1,1,1])

metrics.silhouette\_score(x,y,metric='euclidean')

10 topics model

title = ['Beijing lifts red alert with smog set to disperse',

'Source of major pollutant in China smog revealed',

'Xi stresses clean energy use to reduce smoggy days',

'New Zealand university to open innovation center in China',

'School combines soccer with kung fu to train players',

'NASA bets big on private sector to put humans on Mars',

'Why A Tornado-Damaged Facility In New Orleans Is Critical To NASA',

'NASA Plans to Send This Robot Lander to Look for Alien Life on Europa']

10.1 latent semantic indexing/analysis(LSI/LSA)

import sklearn

from sklearn.decomposition import TruncatedSVD

from sklearn.feature\_extraction.text import TfidfVectorizer

#计算TF-IDF

vectorizer = TfidfVectorizer()

corpus\_tfidf=vectorizer.fit\_transform(title).toarray()

#build 二维语义空间

svd = TruncatedSVD(n\_components=2)

svd.fit(corpus\_tfidf)

svd.fit\_transform(corpus\_tfidf)

10.2 latent dirichlet allocation(LDA)

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n\_topics=2)

vectorizer = TfidfVectorizer()

corpus\_tfidf=vectorizer.fit\_transform(title).toarray()

lda.fit\_transform(corpus\_tfidf)

二 example

1，read data

import pandas as pd

df\_news = pd.read\_table('sohu\_train.txt', sep='\t', header=None,names=['category','content'],encoding='utf-8')

df\_news=df\_news.dropna()

df\_news.head()

df\_news.shape

content=df\_news.content.values.tolist()

print(content[1000])

2, jieba分词

import pandas as pd

import jieba

content\_S=[]

for line in content:

current\_segment=jieba.lcut(line)

if len(current\_segment)>1 and current\_segment !='\r\n':

content\_S.append(current\_segment)

content\_S[1000]

df\_content=pd.DataFrame({'content\_S':content\_S})

df\_content.head()

3， 清除stopwords

stopwords=pd.read\_csv('stopwords.txt',index\_col=False,sep='\t',quoting=3,names=['stopword'],encoding='utf-8')

stopwords.head()

def drop\_stopwords(contents,stopwords):

contents\_clean=[]

all\_words=[]

for line in contents:

line\_clean=[]

for word in line:

if word in stopwords:

continue

line\_clean.append(word)

all\_words.append(str(word))

contents\_clean.append(line\_clean)

return contents\_clean,all\_words

contents=df\_content.content\_S.values.tolist()

stopwords=stopwords.stopword.values.tolist()

contents\_clean,all\_words=drop\_stopwords(contents,stopwords)

df\_content=pd.DataFrame({'contents\_clean':contents\_clean})

df\_content.head()

4,计算TF-IDF

import gensim

from gensim import corpora, models, similarities

dictionary=corpora.Dictionary(contents\_clean)

corpus=[dictionary.doc2bow(sentence) for sentence in contents\_clean]

5,计算similarities

5.1 输出第一篇文档与其他文档之间的相似度

from gensim.similarities import MatrixSimilarity

sim\_index = MatrixSimilarity(corpus)

sim\_index[corpus[0]]

5.2 输出本文档序号与相似度

print (list(enumerate(sim\_index[corpus[0]])))

5.3 按相似度从大到小排序

sort\_sims = sorted(enumerate(sim\_index[corpus[0]]), key=lambda item: item[1],reverse=True)

5.4 输出最相似度前10个文档内容，查看检索效果

for j in [i[0] for i in sort\_sims[0:10]]:

print (j,"\n",content[j])

6 把5中cluster后度文本copy到一个空白到txt文件中，并储存为‘lad.txt'

7 提取词频最高的前10个关键词及其词频率

from gensim.models.ldamodel import LdaModel

lda=gensim.models.ldamodel.LdaModel(corpus=corpus,id2word=dictionary,num\_topics=20)

for topic in lda.print\_topics(num\_topics=20,num\_words=5):

print(topic[1])

print(lda.print\_topic(2,topn=5))

三 应用：长沙市政府工作报告中关于环境管制的关键词频

# 载入程序包

import pandas as pd

import jieba

from collections import Counter

import os

# 指定工作路径

mainpath = '/Users/hym0509/Desktop/out/'

os.chdir(mainpath)

Step1: 读取原始语料

def getFileList(path):

'''

从path中读取文件列表

参数

---

path: str,文件夹路径

'''

path = str(path)

a = os.listdir(path)

filelist = [ x for x in a if os.path.isfile( path + x ) ]

return filelist

file=open(r'/Users/hym0509/Desktop/out/长沙2019.txt','r', encoding='UTF-8').read()

# 参数'r'表示只读，encoding='UTF-8'表示编码为UTF-8

print(type(file))

print(file)

Step2: 进行分词

# 载入结巴中文分词程序包

import jieba

# 读取之前搜集的中文停用词文档

stopfile=open(r'/Users/hym0509/Desktop/out/stopwords.txt', 'r', encoding='UTF-8').read()

# 将中文停用词文档读取为python列表

stopfile = stopfile.replace(" ","")

stoplist = stopfile.split('\n')

# 分词，并按照 if 后的条件对分词进行筛选

words = [x for x in jieba.cut(file) if len(x) >= 2 and x not in stoplist]

# 统计出现频次最高的十个词 (与本例子无关)

top10=Counter(words).most\_common(10)

import json

print(json.dumps(top10, ensure\_ascii=False))

Step3: 统计重点词汇的频次和在全文的比重

通过以上步骤，我得到了一个经过筛选后的分词列表words，列表中的元素为词语及其出现的频次。

接下来，在words中统计我关注的重点词汇的出现频次和比重。

首先，定义由重点词汇构成的词语列表，然后分别统计这些词语在words中的频次，结果保存在一个例表中：

# 以以下8个词为例

keywords = ['环境保护', '环保', '污染', '能耗', '绿色', '污水', '废气', 'pm2.5']

# 将列表words转为特定的计数格式

b = Counter(words)

# 提取重点词汇的频次

wordsfreq = [b[x] for x in keywords]

totalfreq = sum(wordsfreq)

# 所有词语的总数

s = sum(b.values())

# 计算比重

weight = totalfreq/s

# 打印结果

print(wordsfreq)

print(totalfreq)

print(weight)