社交平台有害信息侦测

一、课前准备

- 了解TF-IDF的基本思想
- 了解jieba分词的使用方式

二、课堂主题

- 本节课主要讲授社交平台有害信息侦测数据的处理方法及项目制作。

三、课堂目标

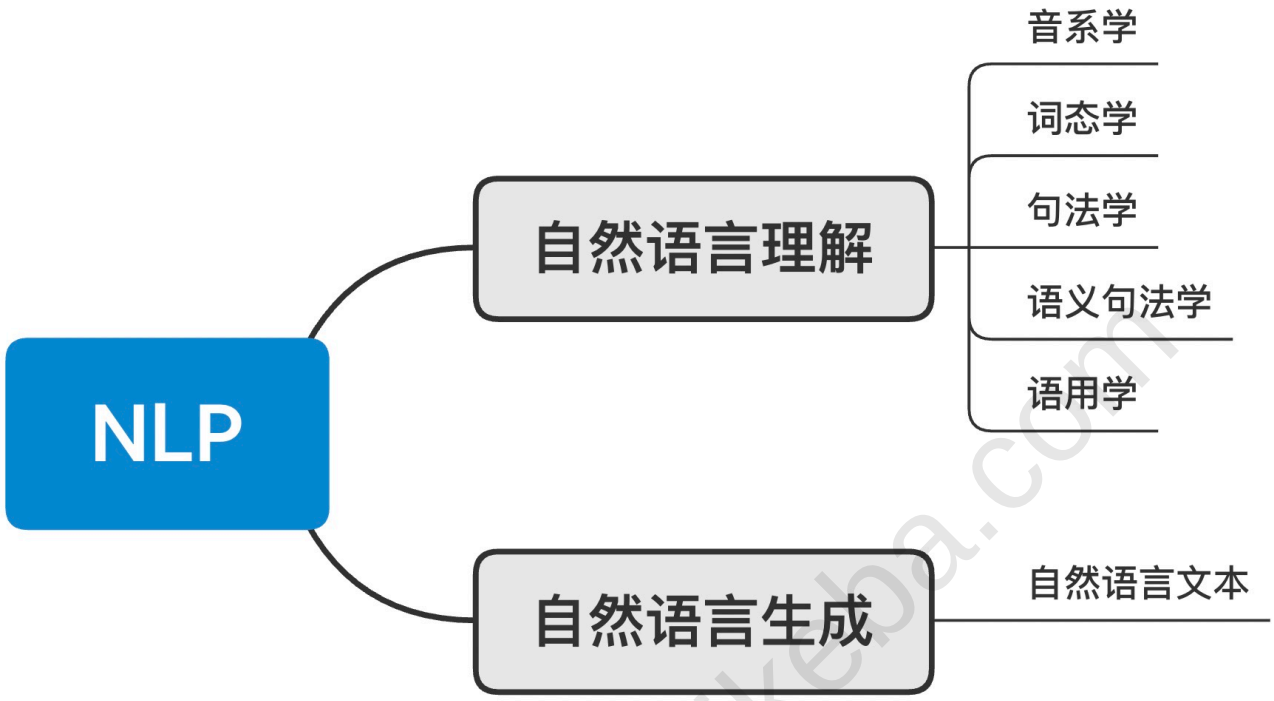
- 了解处理文本数据的方法
- 能够掌握中英文句子的处理方法
- 能够学会侦测评论信息中的有害信息

四、知识要点

4.1 NLP简介

NLP（Natural Language Processing）自然语言处理

4.1.1 NLP的基本分类



- 音系学：语言中发音的系统化组织
- 词态学：研究单词构成及相互之间的关系
- 句法学：指定文本的哪部分是语法正确的
- 语义学：给定文本的含义是什么
- 语用学：文本的目的是什么

4.1.2 NLP的研究任务

- 机器翻译：计算机具备将一种语言翻译成另一种语言的能力。
- 情感分析：计算机能够判断用户评论是否积极。
- 智能问答：计算机能够正确回答输入问题。
- 文摘生成：计算机能够准确归纳、总结并产生文本摘要。
- 文本分类：计算机能够采集各种文章，进行主题分析，从而进行自动分类。
- 舆论分析：计算机能够判断目前舆论的导向。
- 知识图谱：知识点相互连接而成的语义网络。

4.2 TF-IDF回顾

TF-IDF是一种统计方法，用来评估某个词语（或短语）对于一篇文章（或一个句子）的重要程度；而重要性高的词语一般与文章中心思想的相关程度也较高，因此可以作为区分或挖掘文本内容的标签。

TF-IDF的主要思想是：如果某个词语在一篇文章中出现的频率高，并且在其他文章中出现的频率相对较低，则这个词语或者短语对于这篇文章而言的重要性就很高，且具有很好的类别区分能力（将这篇文章与其他文章区分开），因此可以借助这类的词语做文本分类或文本挖掘。

TF

词频（TF）是词语出现的次数除以该文章的词语总数。

- 比如一篇文章的词语总数是100个，而词语“鸡蛋”出现了3次，那么“鸡蛋”这一词在该文章中的词频就是3/100=0.03。

IDF

IDF的公式：$IDF(x) = \log \frac{N}{N(x)}$

IDF公式平滑后：$IDF(x) = \log \frac{N+1}{N(x)+1} + 1$

逆文本频率（IDF）的方法是文章集的文章总数，除以出现“鸡蛋”一词的文章数。

- 所以，如果“鸡蛋”一词在1,000份文件出现过，而文件总数是1000万份的话，其逆向文件频率就是4。

（$\lg(\frac{10,000,000}{1,000}) = 4$，即10的4次方取log值是4 ）

- 最后，“鸡蛋”的TF-IDF值为0.03 * 4=0.12。

TF-IDF

TF−IDF(x)=TF(x)∗IDF(x)

TF-IDF的基本思想是：词语的重要性与它在文件中出现的次数成正比，但同时会随着它在语料库中出现的频率成反比下降。 但无论如何，统计每个单词在文档中出现的次数是必要的操作。所以说，TF-IDF也是一种基于 bag-of-word 的方法。

首先我们来做分词，其中比较值得注意的地方是我们设法剔除了其中的标点符号（显然，标点符号不应该成为最终的关键词）。

词袋（bag of word）

词袋模型是最早的以词语为基本处理单元的文本向量化方法。

有一段文本："it is a beautiful day today"

进行分词处理：it/is/a/beautiful/day/today

得到词袋：（"it","is","a","beautiful","day","today"）

但是我们得到的词袋要交给计算机处理，计算机并不能很好的识别英文，只能识别数字，那么我们给每一个词添加一个索引去解决这个问题：

词袋索引：（0,1,2,3,4,5）

对于需要考虑词频的词袋模型我们可以定义成用一个数字来表示索引一个数字来表示词频的形式：[(0,2),(1,2),(2,2).....]

上例是词袋模型的一种表示方法，下面举另外一个例子来表示一下：

给出两个文本：

1：Bob likes to play basketball, Jim likes too.

2：Bob also likes to play football games.

我们可以构造出一个词典：

{1:Bob", 2. "like", 3. "to", 4. "play", 5. "basketball", 6. "also", 7. "football", 8. "games", 9. "Jim", 10. "too"}。

上面的词典中包含了10个单词，对于每个单词都有一个唯一的索引，那么对于两个文本我们有如下的词袋表示方法：

1：[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]

2：[1, 1, 1, 1 ,0, 1, 1, 1, 0, 0]

词袋模型适用的数据类型：离散，高维，稀疏

词袋模型存在的问题：

1、维度灾难，如果一个例子词典中包含10000个单词，那么每个文本需要用10000维的向量表示。

2、无法保留词序信息。

3、存在语义鸿沟的问题。

4.3 TF-IDF的代码实践

4.3.1 手动预处理

```
import nltk
import math
import string

from nltk.corpus import stopwords
from collections import Counter
from nltk.stem.porter import *

from sklearn.feature_extraction.text import TfidfVectorizer

text1 = "Rising crude oil prices mean that global demand is increasing and that can indicate the
global economy is growing.  \
But the price of oil which is sold by the barrel has fallen dramatically because of the corona virus
pandemic \
and it made headlines in history this week \
because at one point, U.S. stock market oil prices were negative. \
On Monday afternoon, oil was at minus 37.63 per barrel. \
What this meant is that there was so much supply in the oil market that the world was running out of
places to store it, \
and oil producers were essentially paying people to take their oil barrels."

text2 = "Python, from the Greek word (πύθων/πύθωνας), is a genus of \
nonvenomous pythons[2] found in Africa and Asia. Currently, 7 species are \
recognised.[2] A member of this genus, P. reticulatus, is among the longest \
snakes known."

text3 = "Crude prices did turn positive again yesterday \
though they were still incredibly low. \
No one knows how long they'll stay so low and \
that's a concern especially for American oil companies \
because they can't make money. \
Industry analysts say several hundred oil companies will file for bankruptcy by the end of next year
\
if prices don't climb back up. \
Countries outside the U.S. that export huge amounts of crude oil have agreed to reduce their
production of it to try to stabilize prices \
but that won't happen until next month."
```

```
def get_tokens(text):
    lowers = text.lower()
    #移除标点符号，使用分词
    remove_punctuation_map = dict(
        (ord(char), None) for char in string.punctuation)
    no_punctuation = lowers.translate(remove_punctuation_map)
    tokens = nltk.word_tokenize(no_punctuation)
    return tokens
```

下面的代码用于测试分词的结果，Counter() 函数用于统计每个单词出现的次数，输出出现次数排名前15位的单词。

```
tokens = get_tokens(text1)
count = Counter(tokens)
print (count.most_common(15))
```

```
[('oil', 7), ('the', 6), ('that', 4), ('is', 4), ('and', 3), ('of', 3), ('was', 3), ('prices', 2),
('global', 2), ('barrel', 2), ('because', 2), ('it', 2), ('in', 2), ('this', 2), ('at', 2)]
```

显然，像 the, a, and 这些词尽管出现的次数很多，但是它们与文档所表述的主题是无关的，所以我们还要去除“词袋”中的“停词”（stopwords），代码如下：

```
tokens = get_tokens(text1)
filtered = [w for w in tokens if not w in stopwords.words('english')]
count = Counter(filtered)
print(count.most_common(15))
```

```
[('oil', 7), ('prices', 2), ('global', 2), ('barrel', 2), ('market', 2), ('rising', 1), ('crude',
1), ('mean', 1), ('demand', 1), ('increasing', 1), ('indicate', 1), ('economy', 1), ('growing', 1),
('price', 1), ('sold', 1)]
```

从下面的输出结果可以发现，之前那些缺乏实际意义的 the, a, and 等词已经被过滤掉了。

但这个结果还是不太理想，像 films, film, filmed 其实都来源于 film，我们不应该把相同词源的每个词形分开统计，而是应该溯源到词源，统计词源出现的次数。

这时就需要用到 Stemming 方法。代码如下：

```
# 进行单词溯源
def stem_tokens(tokens, stemmer):
    stemmed = []
    for item in tokens:
        stemmed.append(stemmer.stem(item))
    return stemmed
```

```
tokens = get_tokens(text1)
filtered = [w for w in tokens if not w in stopwords.words('english')]
stemmer = PorterStemmer()
stemmed = stem_tokens(filtered, stemmer)
```

使用Stemming方法将分词后的结果处理之后，统计计数排在前15位的词语以及词语出现的次数）：

```
count = Counter(stemmed)
print(count.most_common(15))
```

```
[('oil', 7), ('price', 3), ('barrel', 3), ('global', 2), ('market', 2), ('rise', 1), ('crude', 1),
('mean', 1), ('demand', 1), ('increas', 1), ('indic', 1), ('economi', 1), ('grow', 1), ('sold', 1),
('fallen', 1)]
```

4.4 用scikit-learn进行TF-IDF处理

在scikit-learn中，有2种方法进行TF-IDF的预处理。

1. 用CountVectorizer类向量化之后、再调用TfidfTransformer类进行预处理。
2. 直接用TfidfVectorizer完成向量化与TF-IDF预处理。

CountVectorizer + TfidfTransformer = TfidfVectorizer

4.4.1 CountVectorizer+TfidfTransformer的组合

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

corpus = [
    "I went to Germany for work", "That is a company famous in Germany",
    "I like people in Germany", "The work is recording the voice of people"
]

#该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j] 表示第j个词在第i个文本中的词频
vectorizer = CountVectorizer()
#该类会统计每个词语的tf-idf权值
transformer = TfidfTransformer()

tfidf = transformer.fit_transform(
    vectorizer.fit_transform(corpus))  # CountVectorizer+TfidfTransformer的组合

tfidf.todense()  # todense()返回矩阵，由文本内容转化而来的tf-idf权重矩阵
```

```
matrix([[0.        , 0.        , 0.49819711, 0.31799276, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.49819711, 0.        , 0.49819711,
         0.39278432],
        [0.46370919, 0.46370919, 0.        , 0.29597957, 0.36559366,
         0.36559366, 0.        , 0.        , 0.        , 0.        ,
         0.46370919, 0.        , 0.        , 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , 0.39205255, 0.4842629 ,
         0.        , 0.61422608, 0.        , 0.4842629 , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.26480063, 0.        , 0.33586602, 0.26480063, 0.33586602,
         0.        , 0.67173204, 0.        , 0.33586602, 0.        ,
         0.26480063]])
```

```
tfidf.todense().shape
# corpus中4小段文本，文本词语分别一一对应16个特征；权重矩阵的第i行第j列的值即为第i段文本中第j个词语对应的tf-idf权值
```

```
(4, 16)
```

```
vectorizer.vocabulary_ # 词语与列的对应关系
```

```
{'went': 14,
 'to': 12,
 'germany': 3,
 'for': 2,
 'work': 15,
 'that': 10,
 'is': 5,
 'company': 0,
 'famous': 1,
 'in': 4,
 'like': 6,
 'people': 8,
 'the': 11,
 'recording': 9,
 'voice': 13,
 'of': 7}
```

4.4.2 用TfidfVectorizer一步到位

```
from sklearn.feature_extraction.text import TfidfVectorizer

# TfidfVectorizer类会将文本中的词语转换为词频矩阵，并统计每个词语的tf-idf权值，形成tf-idf权重矩阵
tfidf2 = TfidfVectorizer()
re = tfidf2.fit_transform(corpus)
re.todense()       # todense()返回矩阵，由文本内容转化而来的tf-idf权重矩阵
```

```
matrix([[0.        , 0.        , 0.49819711, 0.31799276, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.49819711, 0.        , 0.49819711,
         0.39278432],
        [0.46370919, 0.46370919, 0.        , 0.29597957, 0.36559366,
         0.36559366, 0.        , 0.        , 0.        , 0.        ,
         0.46370919, 0.        , 0.        , 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , 0.39205255, 0.4842629 ,
         0.        , 0.61422608, 0.        , 0.4842629 , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.26480063, 0.        , 0.33586602, 0.26480063, 0.33586602,
         0.        , 0.67173204, 0.        , 0.33586602, 0.        ,
         0.26480063]])
```

```
re.todense().shape
```

```
(4, 16)
```

```
tfidf2.vocabulary_ # 词语与列的对应关系
```

```
{'went': 14,
 'to': 12,
 'germany': 3,
 'for': 2,
 'work': 15,
 'that': 10,
 'is': 5,
 'company': 0,
 'famous': 1,
 'in': 4,
 'like': 6,
 'people': 8,
 'the': 11,
 'recording': 9,
 'voice': 13,
 'of': 7}
```

4.5 中文文本的分词

4.5.1 分词方法

规则分词

基于规则的分词方法是一种机械分词方法，主要是通过维护词典，在切分语句时，将语句的每个字符与词表中的词进行逐一匹配，找到则切分，否则不予切分。

按照匹配切分的方式，主要有正向最大匹配法，逆向最大匹配法以及双向最大匹配法。

正向最大匹配法

正向最大匹配法的基本思想是：假定分词词典中的最长词有i个汉字，则用被处理文档的当前字串中的前i个字作为匹配字段，查找字典，若字典中存在这样的一个i字词，则匹配成功，匹配字段被作为一个词切分出来。如果词典中找不到这样一个i字词，则匹配失败，将匹配字段中的最后一个字去掉，对剩下的字符串重新进行匹配处理。如此进行下去，直到匹配成功，即切分出一个词或剩余字串的长度为零为止。这样就完成了一轮匹配，然后取下一个i字字串进行匹配处理，直到文档被扫描完为止。

逆向最大匹配法

逆向最大匹配法的基本原理与正向相同，不同的是分词切分的方向与正向最大匹配相反，逆向最大匹配从被处理文档的末端开始扫描匹配，每次取最末端的i个字符作为匹配字段（i为词典中最长词数）作为匹配字段，若匹配失败，则去掉匹配字段最前面一个字，继续匹配。

双向最大匹配法

双向最大匹配法是将正向最大匹配法得到的分词结果和逆向最大匹配法得到的结果进行比较，然后按照最大匹配原则，选取词数切分最少的作为结果。

4.5.2 jieba分词器的分词模式

jieba分词器提供了三种常用的分词模式

> 1、精确模式：将句子按照最精确的方法进行切分，适合用于进行文本分析；

> 2、全模式：将句子当中所有可以成词的词语都扫描出来，分词速度很快但容易产生歧义；

> 3、搜索引擎模式：在精确模式分词的基础上，将长的句子再次进行切分，提高召回率，适用于搜索引擎的分词。

注：jieba也支持对繁体字进行分词。

```
import jieba
ex = '南京市长江大桥'

# 全分词模式
all_cut = jieba.cut(ex, cut_all=True)
# 精确分词模式
precise_cut = jieba.cut(ex, cut_all=False)
# 当我们省略掉cut_all参数时，cut_all默认值为False，此时分词模式为精确分词
default_precise_cut = jieba.cut(ex)
# 搜索引擎模式
search_cut = jieba.cut_for_search(ex)

print("全分词： ", "/".join(all_cut))
print("精确分词： ", "/".join(precise_cut))
print("默认精确分词： ", "/".join(default_precise_cut))
print("搜索分词： ", "/".join(search_cut))
```

```
Building prefix dict from the default dictionary ...
Dumping model to file cache /var/folders/t_/9xpgv1qs5pg7k2qqk4v_6xjh0000gn/T/jieba.cache
Loading model cost 0.701 seconds.
Prefix dict has been built successfully.
```

```
全分词：  南京/南京市/京市/市长/长江/长江大桥/大桥
精确分词：  南京市/长江大桥
默认精确分词：  南京市/长江大桥
搜索分词：  南京/京市/南京市/长江/大桥/长江大桥
```

4.5.3 jieba分词器字典的补充

jieba分词器有两种补充字典的形式，一种是自定义文件导入的静态补充，一种是利用其内置函数的动态补充。

- 静态补充：

我们可以自定义词典，以便包含jieba词典中没有的词（虽然jieba有新词识别能力，但是添加自定义词典可以提高准确率）

添加格式
词语　词频（可省略）　词性（可省略）

我们可以按照上面三个属性去添加新的词语，属性之间用一个空格分开即可。

添加方法函数

`jieba.load_userdict(file_name)　　# file_name为我们要添加的词典的路径`

假设我们现在有文件add_words.txt为要添加的词典，词典中设定的内容如下，我们用全分词模式来验证结果。

```
长江大 5
江大桥 3 nz
南京市长江 2
```

```
import jieba
ex = '南京市长江大桥'
print("更新前的全分词结果:  ", "/".join(jieba.cut(ex, cut_all=True)))
jieba.load_userdict("add_words.txt")
print("更新词典后的全分词结果:  ", "/".join(jieba.cut(ex, cut_all=True)))
```

- 动态补充:

我们可以使用jieba.add_word()和jieba.del_word()两种函数来动态的添加或者删除词语，其中add_word()可以添加词频和词性两种参数。

```
import jieba
ex = '南京市长江大桥'
print("更新前的全分词结果:  ", "/".join(jieba.cut(ex, cut_all=True)))
jieba.add_word("南京市长江")
jieba.add_word("南京市长江", freq=5, tag='nz')
jieba.del_word("南京")
print("更新词典后的全分词结果:  ", "/".join(jieba.cut(ex, cut_all=True)))
```

4.5.4 使用聚类进行中文信息监测

导入相关的包

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
#from data_utils import *
import jieba
import matplotlib.pyplot as plt
import re, string
```

加载语料

代码中定义了两种分词方式

- 一种是单纯的使用bi-gram分词,
- 一种是使用jieba进行中文分词,

根据效果选择使用哪种分词方式。

加载语料的时候把标点符号去掉，这对于文本聚类几乎没有作用。

还可以去掉自己定义的一些类似“你”、“我”、“他”等大量口语化中均会出现的停止词，这类词语往往对聚类也起不到作用。

```
import re
punctuation = '!,;:?"\''


def removePunctuation(text):
    # 使用正则把标点替换掉
    text = re.sub(r'[{}]+'.format(punctuation), '', text)
    return text.strip().lower()
```

```
# bigram分词
segment_bigram = lambda text: " ".join([
    word + text[idx + 1] for idx, word in enumerate(text)
    if idx < len(text) - 1
])
```

```
# jieba中文分词
segment_jieba = lambda text: " ".join(jieba.cut(text))

corpus = []
i = 0
with open("text.txt", "r", encoding="utf-8") as f:
    for line in f:
        if i < 1000:
            # 去掉标点符号
            corpus.append(segment_jieba(removePunctuation(line.strip())))
            i += 1
```

```
Building prefix dict from the default dictionary ...
Dumping model to file cache /var/folders/t_/9xpgv1qs5pg7k2qqk4v_6xjh0000gn/T/jieba.cache
Loading model cost 0.680 seconds.
Prefix dict has been built successfully.
```

计算tf-idf设为权重

```
# 计算tf-idf设为权重
vectorizer = CountVectorizer()
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))

#获取词袋模型中所有的词语特征，如果特征数量非常多的情况下需要按照权重降维
word = vectorizer.get_feature_names()
print("word feature length: {}".format(len(word)))    #词语特征的数量

#导出权重，到这边就实现了将文字向量化的过程，矩阵中的每一行就是一条文本记录的向量表示
tfidf_weight = tfidf.toarray()
```

```
word feature length: 47042
```

文本向量化的矩阵tfidf_weight的词语特征数量为47042个，数量太大，需要对tfidf_weight进行降维。

采用阶段奇异值分解SVD降维

TruncatedSVD与PCA相比，这种方法可以有效实现稀疏矩阵的降维；而PCA不能。

另外，truncated SVD作用于sklearn.feature_extraction.text向量化后返回的TF-IDF矩阵，这种情况下就是LSA/LSI (即潜在语义分析)。

```
from sklearn.decomposition import TruncatedSVD

n_pick_topics = 12              # 设定主题数为3
lsa = TruncatedSVD(n_pick_topics)
X2 = lsa.fit_transform(tfidf_weight)
```

```
X2.shape
```

```
(1000, 12)
```

```
# PCA降维
# from sklearn.decomposition import PCA

# pca=PCA()
# new=pca.fit_transform(tfidf)
# ratio=pca.explained_variance_ratio_

# PCA不能实现稀疏矩阵的降维
```

```
#对向量进行聚类
# 指定分成7个类
kmeans = KMeans(n_clusters=6)
kmeans.fit(X2)

# 打印出各个族的中心点
print(kmeans.cluster_centers_)
for index, label in enumerate(kmeans.labels_, 1):
    print("index: {}, label: {}".format(index, label))

# 样本距其最近的聚类中心的平方距离之和，用来评判分类的准确度，值越小越好
# k-means的超参数n_clusters可以通过该值来评估
print("inertia: {}".format(kmeans.inertia_))
```

可视化

前面将文本向量化之后，每个文档的维度非常高，进行可视化之前需要对其降维。

降维算法也有很多，这里使用T-SNE算法，其优点就是准确度比较高，但是耗时比较长，如果接受不了耗时长，可以使用PCA算法。

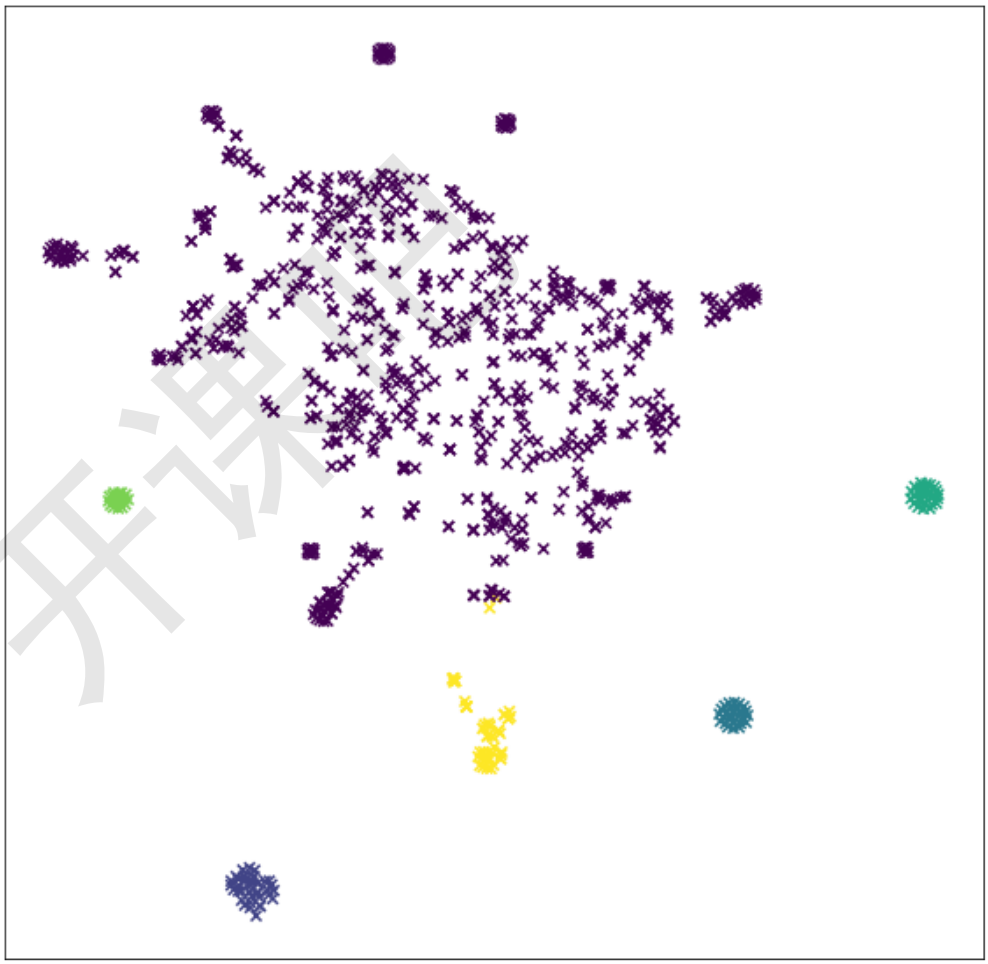
但注意，无论是T-SNE还是PCA算法，均不能用于稀疏矩阵的降维，例如本文中tfidf/tfidf_weight的稀疏矩阵，我们只能用TruncatedSVD来降维到12个维度，之后可视化的时候再用T-SNE或者PCA算法降到2维。

```
# 使用T-SNE算法，对权重进行降维，准确度比PCA算法高，但是耗时长
tsne = TSNE(n_components=2)
decomposition_data = tsne.fit_transform(X2)

x = []
y = []

for i in decomposition_data:
    x.append(i[0])
    y.append(i[1])

fig = plt.figure(figsize=(10, 10))
ax = plt.axes()
plt.scatter(x, y, c=kmeans.labels_, marker="x")
plt.xticks(())
plt.yticks(())
plt.show()
#plt.savefig('./sample.png', aspect=1)
```



结果呈现

聚类之后根据原文档中的内容判断聚类后的类别为有害/非有害信息。

4.6 英文文本的分词

4.6.1 标准英文句子的分词方式

```
# 使用空格作为分隔符进行分词
text = "I like machine learning."
lst = text.split(' ')
lst
```

```
['I', 'like', 'machine', 'learning.']
```

4.6.2 NLTK分词

语言处理任务	NLTK模块	功能描述
获取和处理语料库	nltk.corpus	语料库和词典的标准化接口
字符串处理	nltk.tokenize, nltk.stem	分词，句子分解提取主干
搭配发现	nltk.collocations	t-检验，卡方，点互信息PMI
词性标识符	nltk.tag	n-gram, backoff, Brill, HMM, TnT
分类	nltk.classify, nltk.cluster	决策树，最大熵，贝叶斯，EM，k-means
分块	nltk.chunk	正则表达式，n-gram，命名实体
解析	nltk.parse	图表，基于特征，一致性，概率，依赖
语义解释	nltk.sem, nltk.inference	λ演算，一阶逻辑，模型检验
指标评测	nltk.metrics	精度，召回率，协议系数
概率与估计	nltk.probability	频率分布，平滑概率分布
应用	nltk.app nltk.chat	图形化的关键词排序，分析器，WordNet查看器，聊天机器人
语言学领域的工作	nltk.toolbox	处理SIL工具箱格式的数据

4.7 社交平台有害信息侦测

4.7.1 项目背景

社交媒体日益成为人们进行网络活动的主要平台,网络有害信息的传播变得更多样、更快、渠道更广，有害信息的侦测成为净化社交媒体的重要方式。

本项目数据集包括某社交平台的用户评论数据与对应是否属于各类有害信息的标签。

基于TF-IDF算法将评论文本内容转化为特征向量（权重矩阵），并基于训练集数据训练逻辑回归模型，来判断测试集的评论中是否中包含有害信息。

4.7.2 核心技术

```
数据分析：数据的分组统计
数据预处理：标签编码、中文分词库jieba的应用
特征工程：基于TF-IDF算法将文本内容向量化
模型选择：逻辑回归模型、交叉验证
```

4.7.3 使用场景

能够帮助高效地识别社交类平台与应用中的有害信息，为社交平台的规范化运营提供有效的支持。

4.7.4 职业发展

从事社交媒体、互联网金融等的数据分析、数据挖掘等相关工作。

4.7.5 数据集

train.csv文件中每条记录对应某社交平台上的一条评论内容。

这些评论内容均已经被标注了所属的类别，包括了toxic(有害)\severe_toxic(严重有害)\obscene(淫秽的)\threat(恐吓威胁)\insult(侮辱性的)\identity_hate(种族歧视、种族仇恨)；而正常的评论对应的这六个标签值为0。

现需要基于TF-IDF算法将评论文本内容转化为特征向量（权重矩阵），并基于训练集数据（包括了评论内容与对应的有害标签）训练逻辑回归模型，来判断测试集的评论中是否中包含有害信息。

4.7.6 项目实战

```
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from scipy.sparse import hstack

train = pd.read_csv('3.0train.csv').fillna(' ')  #训练集
test = pd.read_csv('3.0test.csv').fillna(' ')    #测试集
train=train.iloc[:10000]
test=test.iloc[:8000]

class_names = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
```

```
train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

大致可以看到，comment_text是社交平台的评论内容(X)，之后'toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate'分别是有害标签(y)

```
train_text = train['comment_text']
test_text = test['comment_text']

all_text = pd.concat([train_text, test_text])
```

```
word_vectorizer = TfidfVectorizer(
    sublinear_tf=True,
    strip_accents='unicode',  #  在预处理步骤中去除编码规则
    analyzer='word',  #  analyzer='word'，词频矩阵构建过程会默认过滤所有的单字token
    token_pattern=r'\w{1,}',
    stop_words='english',  # 直接英语内建的停用词列表
)
word_vectorizer.fit(all_text)
train_word_features = word_vectorizer.transform(train_text)
test_word_features = word_vectorizer.transform(test_text)
```

```
char_vectorizer = TfidfVectorizer(
    sublinear_tf=True,
    strip_accents='unicode',
    analyzer='char',  # 自动按照单字进行分隔统计词频
    stop_words='english')
char_vectorizer.fit(all_text)
train_char_features = char_vectorizer.transform(train_text)
test_char_features = char_vectorizer.transform(test_text)
```

```
/Users/seven/opt/anaconda3/lib/python3.7/site-packages/sklearn/feature_extraction/text.py:497: UserWarning:
The parameter 'stop_words' will not be used since 'analyzer' != 'word'
  warnings.warn("The parameter 'stop_words' will not be used"
```

创建TfidfVectorizer实例时，有一个默认参数analyzer='word'，在该参数作用下，词频矩阵构建过程会默认过滤所有的单字token，所以例如'学 习 知 识'以空格分隔、被识别为全是单字，全被过滤，输出为empty vocabulary。

如果想针对单字进行tfidf计算，可以加上参数vectorizer = TfidfVectorizer(analyzer='char')，此时，输入字符串无需做空格分隔，TfidfVectorizer会自动按照单字进行分隔统计词频。

```
# 基于“字”与基于“词”的tfidf计算结果的拼接，目的是将相同文本、过滤单字的转化结果与以单字分隔的转化结果拼接起来，更准确
# 由于直接通过tfidf算法转化的向量矩阵是稀疏矩阵，所以需要用scipy.sparse.hstack将矩阵按照列进行拼接
train_features = hstack([train_char_features, train_word_features])
test_features = hstack([test_char_features, test_word_features])

scores = []
test_set = pd.DataFrame.from_dict({'id': test['id']})

# 注意由于现在有多个类别，而我们逻辑回归本质上做的是二元的（某一类下的正例vs.某一类下的负例），
# 因此我们就分别取六类中的一类，做逻辑回归
for class_name in class_names:
    train_target = train[class_name]
    classifier = LogisticRegression(C=0.1, solver='sag')
    # 3折交叉验证，评估标准使用auc的值，计算交叉验证的分数
    cv_score = np.mean(cross_val_score(classifier, train_features, train_target, cv=3, scoring='roc_auc'))
    scores.append(cv_score)
    print('CV score for class {} is {}'.format(class_name, cv_score))

    # 训练模型，并对测试集的文本向量预测其标签值
    classifier.fit(train_features, train_target)
    test_set[class_name] = classifier.predict_proba(test_features)[:, 1]

print('Total CV score is {}'.format(np.mean(scores)))

test_set.to_csv('test_set.csv', index=False)
```

```
CV score for class toxic is 0.8752891426843176
CV score for class severe_toxic is 0.9626593831933725
CV score for class obscene is 0.9063087785745129
CV score for class threat is 0.8999893073376682
CV score for class insult is 0.8949871151251555
CV score for class identity_hate is 0.8841517789727069
Total CV score is 0.9038975843146223
```

五、总结

- TF-IDF的概念
- TF-IDF在python中的两种使用方法
- 中文分词和英文分词
- 社交平台有害信息侦测

六、作业

总结对于非结构化数据（中英文文本）的处理方法。