登堂入室——文本分析实现新闻推荐

案例背景

当前，很多网站提供新闻列表，可以令用户方便的查阅自己感兴趣的信息。

然而，用户浏览新闻，也许仅仅只是随便看到一条。网站的资源非常丰富，可能收藏很多用户感兴趣的其他新闻，但是受限于用户对新闻了解的广泛度，很多新闻用户未能发现，网站也因此错误了很多吸引用户的机会。

任务说明

我们的任务在于，可以根据用户浏览过的新闻，去建立模型，进而可以根据浏览的新闻，与网站现有的其他新闻进行匹配，自动推断出用户最可能感兴趣的新闻，从而达到吸引顾客，防止顾客流失，增加网站流量等目的。

扩展联想

该实现可以用于其他推荐的场景，而不限于新闻推荐。例如，音乐，图书，广告等。

数据集描述

数据集采用搜狗2012年6月-7月全网新闻数据。该数据集为xml文件格式类型。格式如下：

```
<doc>
<url>新闻链接</url>
<docno>新闻编号</docno>
<contenttitle>新闻标题</contenttitle>
<content>新闻内容</content>
</doc>
```

程序实现

前期准备

本案例需要使用3个库（Anaconda没有预安装），分别为：

- wordcloud
- jieba
- gensim

可以使用如下命令进行安装:

```
pip install wordcloud
pip install jieba
pip install gensim
```

导入相关的库

```
import numpy as np
import pandas as pd
import re
import warnings

warnings.filterwarnings("ignore")
```

原始数据的处理

数据集是一个xml类型的文件，我们不能使用pandas直接进行加载。
我们读取原始的数据集，使用正则表达式提取<contenttitle>与</contenttitle>中的标题信息。并将信息输入到另外一个文档中。

```
re_obj = re.compile(r"<contenttitle>(.+?)</contenttitle>")
re_obj2 = re.compile(r"<content>(.+?)</content>")
with open("news_tensite_xml.dat", encoding="ANSI") as f, open("news.dat", "wt",
encoding="ANSI") as f2:
    for line in f:
        match = re_obj.match(line)
        if match:
            f2.write(match.group(1))
            line2 = f.readline()
            match2 = re_obj2.match(line2)
            if match2:
                f2.write(match2.group(1))
            f2.write("\n")
```

加载数据集

加载数据集，并查看数据的基本信息。

```
news = pd.read_csv(r"news.dat", header=None, names=["title"], encoding="ANSI")

display(news.shape)
display(news.head())
# news.iloc[0].tolist()
```

数据清洗

缺失值处理

```
news.isnull().sum()
```

重复值处理

```
# news.duplicated().sum()
# news[news.duplicated()]
news.drop_duplicates(inplace=True)
display(news.shape)
```

去特殊字符，分词，去除停用词

分词

分词是将连续的文本，分割成语义合理的若干词汇序列。我们可以通过jieba来实现分词的功能。

```
import jieba

s = "今天，外面下了一场很大的雨。@#￥#@。"
# jieba.cut(s)
jieba.lcut(s)
```

我们发现，分词会保留标点符号与特殊字符，因此，我们可以在分词前，先将这些内容去除。

去除停用词

“今天，外面下了一场很大的雨。”最能体现这句话核心含义的词语是什么?
A 了 的
B 今天 雨
C 下 雨
D 外面 很大

停用词，指的是在我们语句中大量出现，但却对语义分析没有帮助的词。对于这样的词汇，我们通常可以将其删除，这样的好处在于：

1. 可以降低存储空间消耗。
2. 可以减少计算时间消耗。

综合处理

我们将之前的步骤进行综合处理。

```
re_obj = re.compile(r"[!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~——！，。？、￥…（）：【】
《》‘’“”\s\ue40c]+")

# 获取停用词列表
def get_stopword():
    s = set()
    with open("stopword.txt", encoding="UTF-8") as f:
        for line in f:
            s.add(line.strip())
    return s

stopword = get_stopword()

# 清洗文本数据
def clear(text):
    return re_obj.sub("", text)

# 进行分词的函数。
def cut_word(text):
    return jieba.cut(text)

# 去掉停用词函数。
def remove_stopword(words):
    return [word for word in words if word not in stopword]
```

```
def preprocess(text):
    text = clear(text)
    word_iter = cut_word(text)
    word_list = remove_stopword(word_iter)
    return word_list

# 因为执行时间较长，这里，我们只取前1000条新闻。
# news = news.iloc[:1000]
news = news.iloc[:10]
news["title"] = news["title"].apply(preprocess)

ord("1"), ord("1")
```

文本分析

词云图统计

```
import wordcloud
import matplotlib as mpl
import matplotlib.pyplot as plt

import scipy
from itertools import chain

mpl.rcParams["font.family"] = "SimHei"
mpl.rcParams["axes.unicode_minus"] = False

wc = wordcloud.WordCloud(font_path=r"C:/Windows/Fonts/STFANGSO.ttf", width=800, height=600)
li = news["title"].tolist()
li = list(chain.from_iterable(li))
join_words = " ".join(li)
img = wc.generate(join_words)
plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.axis('off')
wc.to_file("wordcloud.png")
```

此外，我们还可以使用指定的图片作为背景，生成词云图。

```
wc = wordcloud.WordCloud(font_path=r"C:/Windows/Fonts/STFANGSO.ttf",
mask=scipy.misc.imread("map.jpg"))
img = wc.generate(join_words)
plt.figure(figsize=(15, 10))
plt.imshow(img)
plt.axis('off')
```

数据建模

如果需要实现新闻相似度的匹配，我们可以对两条新闻进行分词处理，然后分别比较新闻中出现共同词语的个数。这种做法好吗？

A 好

B 不好

```
documents = news["title"].tolist()
print(documents[0])

import logging
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

documents = news["title"].tolist()
# doc2vec训练的语料与word2vec的格式有些不同。doc2vec要求语料为TaggedDocument构成的数组。
# TaggedDocument代表的就是一篇文档，该文档包含若干个词汇。

# TaggedDocument的参数。 doc 一维数组，文档中的词汇。 [index] 表示文档的标签。
documents = [TaggedDocument(doc, [index]) for index, doc in enumerate(documents)]
# documents：用来训练的语料。要求是TaggedDocument构成的数组。
# vector_size：文档进行嵌入的维度。（文档映射到空间当中，向量的维度）。
# window：当前词汇与预测词汇之间的最远距离。
# min_count：考虑最小词频的词汇。（如果词汇低于该词频，则不再考虑该词汇）
# worker：并发的数量。
# epochs：训练的轮数。
model = Doc2Vec(documents, vector_size=32, window=5, min_count=1, workers=4, epochs=40)

# documents[0].words
# documents[0].tags
vector = model.infer_vector(documents[0].words)
sim = model.docvecs.most_similar([vector], topn=5)
sim
```

模型效果评估

```
ranks = []
for doc in documents:
    # doc.words就是一个文档中包含的词汇列表。infer_vector方法的作用是将参数指定的
    # 文档中的词汇列表表示成向量的形式（推断表示）。即如果参数指定的词汇(文档)映射到
    # 当前的模型空间中，应该表示成为怎样的向量。
    inferred_vector = model.infer_vector(doc.words)
    # 根据参数指定的向量，返回当前模型中，与参数向量最相似的向量。
    sims = model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))
    # 查看推断文档的向量与自己比较，相似度排行第几。（理想情况下，应该排在最前面）
    rank = [tag for tag, _ in sims].index(doc.tags[0])
    ranks.append(rank)
```

我们可以输出查看下模型的效果。

```
from collections import Counter

# Counter([1, 2, 3, 1, 2])
print(Counter(ranks))
```

模型预测

```
display(",".join(documents[0].words))
v = model.infer_vector(documents[0].words)
# display(v)
similar = model.docvecs.most_similar([v], topn=3)
for index, score in similar:
    display(score, ",".join(documents[index].words))
```

词嵌入可视化

```
from sklearn.manifold import TSNE

# 进行降维，参数指定降维的维数。
tsne = TSNE(n_components=2)
# 获取所有词汇标签。
y = list(model.wv.vocab.keys())
# 取部分词汇标签，进行可视化展示。
y = y[:30]
# 根据词获取嵌入向量。
X = model.wv[y]
# 进行降维转换。
X_tsne = tsne.fit_transform(X)

X_tsne.shape

plt.figure(figsize=(16, 16))

for i in range(len(X_tsne)):
    plt.scatter(X_tsne[i,0],X_tsne[i,1])
    # 在散点上打上标签（单词的文本）
    plt.annotate(y[i], fontsize=15,
                 xy=(X_tsne[i,0],X_tsne[i,1]),
                 xytext=(5, 2),
                 textcoords='offset points',
                 ha='right',
                 va='bottom')
plt.show()
```