# 文本检索大作业实验报告

王天宇 2000013108

# 文件说明

> dataprocess.ipynb: 进行数据处理。数据处理的具体细节在下面提到

> GUI.py: 在给定的文件基础上完善了CS架构交互的部分。

> ipynb_importer.py: 作业包下发的原文件未作改动

> local_server.py: 后台服务器

> data folder
>
> > all_news.csv/all_cnews.csv: 下发的原数据集
> >
> > keywords.npy: ndarray，每篇文章的关键词（以编号形式储存）
> >
> > pca.npy: 降维之后的tfidf矩阵
> >
> > synonym.txt: 每个词汇的相似词
> >
> > tfidf_matrix.npy: tfidf矩阵
> >
> > vocab.txt: 处理后构建的词典

要运行文本检索器，需要先运行local_server.py启动服务器，然后运行GUI.py启动图形界面，进行检索即可。

上传的文件中已经包含了所有处理好的数据，所以不需要再次运行数据处理程序（该程序运行需要一定时间）。

# 具体工作

## 数据处理

此部分在dataprocess完成。对数据进行读入、去除数字、标点、停用词、对单词进行lemmatize，并将所有单词整理成字典保存。

## 检索排序

实现了CS架构，检索、排序方案已经经过后续优化，将在后面部分陈述。

根据输入的检索词检索文章，并对检索到的文章进行初步排序，并返回。支持1-3个检索词。若找不到相关文章，会给出相应提示。


## 排序优化

首先，对数据进行相应的处理。

- 提取词汇-文章的tf-idf矩阵

| | TF | IDF | TF-IDF |
|---|---|---|---|
| 0 | 0.046154 | 4.351949 | 0.200859 |
| 27 | 0.015385 | 1.998812 | 0.030751 |
| 311 | 0.007692 | 3.193439 | 0.024565 |
| 397 | 0.007692 | 2.902854 | 0.022330 |
| 512 | 0.023077 | 1.540763 | 0.035556 |
| ... | ... | ... | ... |
| 369018 | 0.009804 | 4.246589 | 0.041633 |
| 439945 | 0.009804 | 6.549174 | 0.064208 |
| 449208 | 0.019608 | 5.856027 | 0.114824 |
| 331062 | 0.009804 | 2.572612 | 0.025222 |
| 27732 | 0.009804 | -0.050697 | -0.000497 |

[478568 rows x 7 columns]

- 对tf-idf矩阵进行PCA降维

```
X = np.load('./data/tfidf_matrix.npy')
X = normalize(X)
dims = 1000
pca = PCA(n_components=dims)
Y = pca.fit_transform(X)
Y = normalize(Y)
```

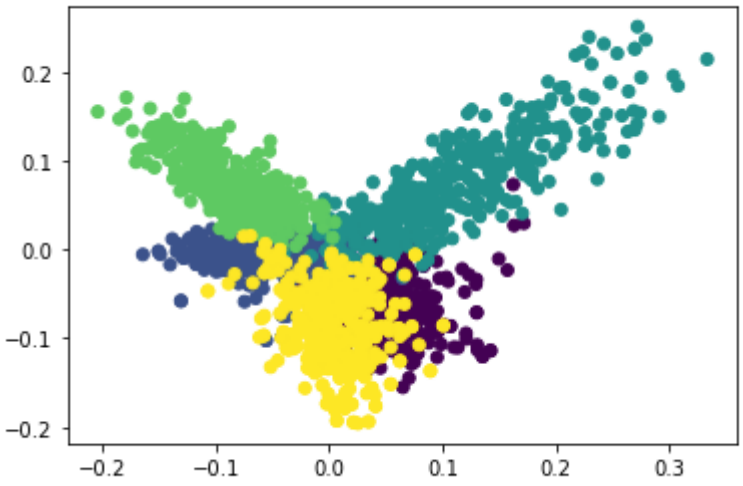
✓ 15.4s

1. 对tfidf矩阵，将检索词对应的文章向量全部叠加之后可以查找最突出的维度，从而实现相关度检索。
2. 依据降维后的矩阵，可以进行文章之间相似度的计算，依此可以计算给定任意文章集合的距离矩阵，从而可以在此矩阵上实现hits算法。
3. 对tfidf矩阵，求取文章向量中最突出的几维，可以实现查找文章关键词的功能。

- 对以上三部分进行权重叠加，最终可以得到当前检索词下的文章相关性排序，从而实现了排序的优化。

# 文章聚类

对tfidf矩阵进行聚类分析。为了方便可视化，将数据降到2维，并计算相关purity




```
    Purity = maxi.sum()/2225
    print("Purity=", Purity)
```

✓ 0.6s

```
nr: 0    505
1    511
2    455
3    369
4    385
Name: 1, dtype: int64
Purity= 0.9402247191011236
```

可以看到，聚类之后的purity达到0.9以上，这说明tfidf向量是有意义的。

# 相似词

对于输入的检索词，扩展检索词对应的相似词。相似词由tfidf矩阵进行词向量的余弦相似度计算得出，取词汇余弦相似度最高的6个。

```
    print(similar_words['rising'])
    print(similar_words['grew'])
    print(similar_words['secure'])
```

✓ 0.1s

```
['barrel', 'oil', 'rising', 'economy', 'prices', 'crude']
['latin', 'china', 'exports', 'growth', 'grew', 'economy']
['bagle', 'cabir', 'program', 'security', 'secure', 'virus']
```

并保存为synonym.txt

# 模糊匹配

在后续进行检索时，相似词对检索的影响是：对于输入的检索词，规定其权重为1；对于检索词的所有相似词，权重定为两者的余弦相似度，并同样参与检索。

```
for word in kwords:
    if word not in self.synonym:
        continue
    for syn in self.synonym[word]:
        id1 = self.word_list.index(word)
        id2 = self.word_list.index(syn)
        vec1 = normalize(self.tfidf[:, id1].reshape(1, -1)).reshape(-1)
        vec2 = normalize(self.tfidf[:, id2].reshape(1, -1)).reshape(-1)
        cos_sim = np.dot(vec1, vec2)
        if id2 not in word_weight:
            word_weight[id2] = cos_sim
        else:
            word_weight[id2] += cos_sim
```

# 加分项

依据tfidf矩阵，对文章进行了关键词提取，并将文章编号-关键词编号的矩阵储存到了keywords.npy中。

```
Dollar gains on Greenspan speech
['recent', 'falls', 'account', 'federal', 'chinese', 'greenspan', 'currency', 'dollar', 'reserve',
'deficit']
Vodafone appoints new Japan boss
['kk', 'technologically', 'customers', 'shiro', 'tetsuro', 'tsusaka', 'japan', 'tsuda', 'morrow',
'vodafone']
US charity anthem is re-released
['grammys', 'richie', 'sang', 'famine', 'springsteen', 'anthem', 'lionel', 'issued', 'dylan',
'recording']
Glaxo aims high after profit fall
['disappointing', 'discontinued', 'wellbutrin', 'paxil', 'depressants', 'allergies', 'profits',
'pipeline', 'obesity', 'garnier']
```

在进行检索的过程中，如果发现检索词在待选文章的关键词集合内，则对其权重有一定的提升。 具体地，设检索到的文章集合为$S$，词汇集合是$W$，定义关键词命中向量$V = (n_1, n_2, ..., n_{|S|}), n_i = |\{w_j | w_j \in W \bigcap s_i\}|$，$s_i$是对应文章的词汇集合。然后对其做标准化，以$\alpha = \frac{1}{3}$的权重参与到最后的计算中。

# 检索和排序的具体过程

对某次检索，定义词汇的权重向量。对于客户端直接检索的词汇，其权重为1。对词汇进行相似词扩展，将相似词的权重定为其与原词汇的余弦相似度。这样得到word_weight向量

依据word_weight向量计算数据集中文章的相关度。定义本次检索的文章相关度为 $\sum_i V_{tfidf_i} \times w_i$ 即词的tfidf向量以word_weight为权重的累加和。

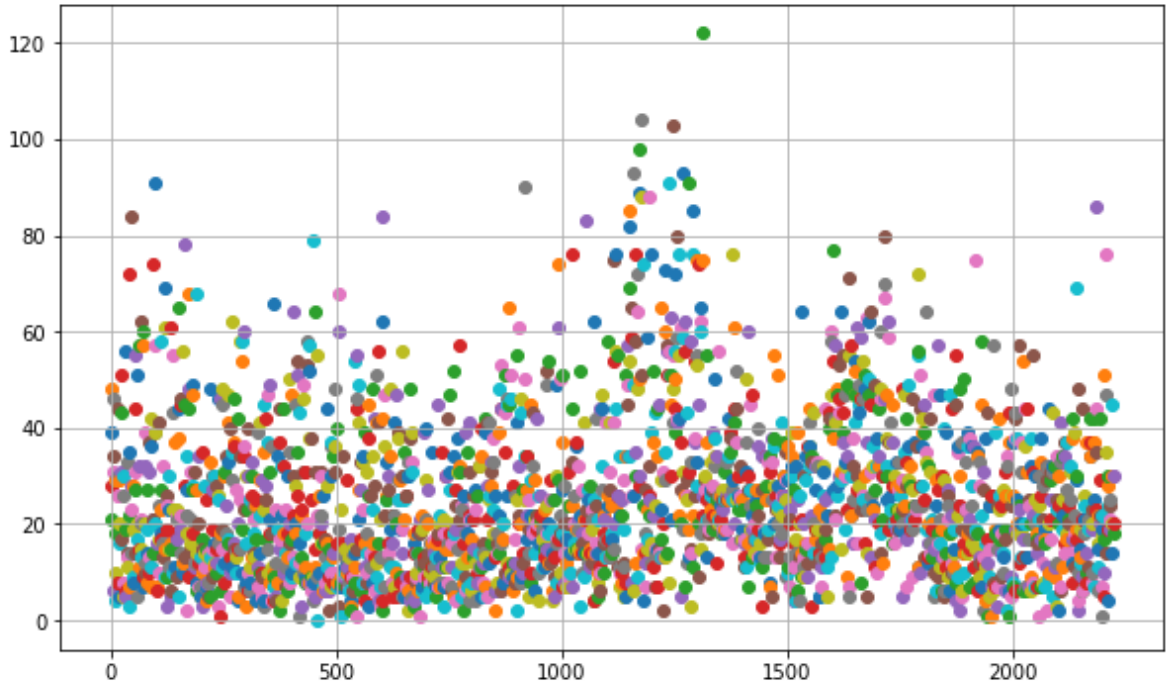
依据此权重，选取相关度较高的一些文章。此处的阈值定为0.1。

检索过程结束，接下来对文章进行排序。排序的权重来着三个部分：

- 刚刚得到的与检索词集合的相关度$weight_1$
- 根据检索词集合与文章关键词集合的重复度，得到的关键词命中权重$weight_2$
- 根据hits算法，计算的文章权重$weight_3$

> hits算法的实现过程：
>
> - 计算文章间的相似度：文章词向量的cosin距离 （归一化）高于的阈值的解读为有关联关系
> - 将与这些诗歌有关联关系的文章加入集合（文章间关联关系构成一对称矩阵）
> - 迭代求该矩阵的主特征向量
> - 按与该向量的cosin结果排降序得到检索结果排序

hits算法的阈值由可视化手动调整：(下图为阈值设定0.1，每篇文章的相关文章数量)




最后的综合权重$w = weight_1 + weight_2 + weight_3$，依据此对检索到的文章进行降序排序。