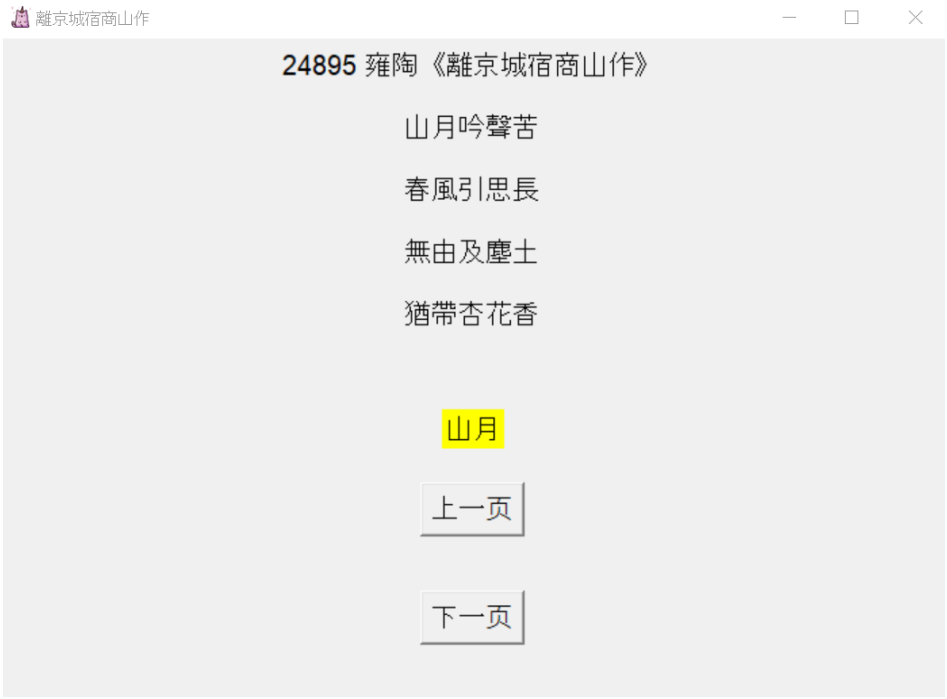
尤俊浩 1900094810 文本检索大作业

实现功能

过滤精度

搜索

高亮&翻页



TF-IDF 特征提取

TF-IDF 矩阵应该是一个每一行为词表，列为词表的上下文的矩阵。每一个格子的值则为 TF-IDF 值。

TF:

首先得出 word_dict，能快速索引每个词在哪个 Poem 里出现，方便后续处理。

```
def get_tf(x):
    context = new_data[new_data.Poem_id.isin(word_dict[x.word])]
    tf = context['word'].value_counts().reset_index().rename(columns={'index':'word','word':'TF'})
    m = pd.merge(WORDLIST,tf,how='left').fillna(0)
    return m.TF.tolist()
```

之后可以对每个词频大于 10 的词，apply(get_tf)，得出他们所有的上下文（context），这样就可以很方便的通过 value_counts 得出每一个词的上下文对应的 TF 向量

IDF:

首先得出每个词对应的所有上下文，然后对上下文进行 groupby，再计算 word.nunique()就能得到每一个词对应的 DF，同时我们已知所有词的数量（每个词都有自己的上下文）就是文章总数，这样就能得到 IDF

	word	Poem_id	context
0	餞	4371	餞
1	餞	4371	唐
2	餞	4371	永昌
3	餞	4371	一作
4	餞	4371	餞
...	...	...	...
21	禁花	39204	濃
22	禁花	39204	樹
23	禁花	39204	禁花
24	禁花	39204	開
25	禁花	39204	後庭

5217518 rows × 3 columns

TF-IDF

	0
0	[18.0, 4.0, 3.0, 8.0, 1.0, 2.0, 1.0, 1.0, 1.0,...
1	[4.0, 191.0, 3.0, 52.0, 2.0, 2.0, 4.0, 6.0, 1....
2	[5.0, 36.0, 1.0, 2474.0, 10.0, 21.0, 18.0, 14....
3	[2.0, 3.0, 1.0, 13.0, 54.0, 4.0, 4.0, 1.0, 1.0...
4	[2.0, 2.0, 1.0, 23.0, 3.0, 69.0, 2.0, 1.0, 1.0...
...	...
2708	[0.0, 0.0, 0.0, 12.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
2709	[0.0, 1.0, 0.0, 6.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2710	[0.0, 0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2711	[0.0, 0.0, 0.0, 68.0, 1.0, 0.0, 0.0, 1.0, 0.0,...
2712	[0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...

```
[8.398551356066518,
 13.793320257839376,
 16.015712679175824,
 14.793320257839376,
 15.60067517989698,
 16.600675179896978,
 17.600675179896978,
 17.600675179896978,
 17.600675179896978,
 13.430750178454668,
 16.015712679175824,
 13.900235461755887,
 16.600675179896978,
 15.278747085009616,
 16.600675179896978,
 14.278747085009616,
```

左图为 TF，右图为与 TF 每一列同长的 IDF 向量，将 TF 每一列与 IDF 相乘即可得到下图的 TF-IDF

```
def mul(x):
    return [i * j for i,j in zip(x,idf)]

TFIDF = tf[0].apply(mul)
display(TFIDF)
```

```
0       [151.17392440919733, 55.173281031357504, 48.04...
1       [33.59420542426607, 2634.524169247321, 48.0471...
2       [41.99275678033259, 496.55952928221757, 16.015...
3       [16.797102712133036, 41.37996077351813, 16.015...
4       [16.797102712133036, 27.586640515678752, 16.01...
                              ...
2708    [0.0, 0.0, 0.0, 177.5198430940725, 0.0, 0.0, 0...
2709    [0.0, 13.793320257839376, 0.0, 88.759921547036...
2710    [0.0, 0.0, 0.0, 59.173281031357504, 0.0, 0.0, ...
2711    [0.0, 0.0, 0.0, 1005.9457775330776, 15.6006751...
2712    [0.0, 0.0, 0.0, 29.586640515678752, 0.0, 0.0, ...
Name: 0, Length: 2713, dtype: object
```

近义词挖掘算法

把刚刚得到的词-上下文表转换成二维矩阵，再利用 scipy 库转成稀疏矩阵，节省内存，加速计算。

```
matrix([[1.        , 0.20018244, 0.37335952, ..., 0.08807629, 0.09913997,
         0.07081448],
        [0.20018244, 1.        , 0.31829516, ..., 0.11803708, 0.18797499,
         0.06557388],
        [0.37335952, 0.31829516, 1.        , ..., 0.21995674, 0.40881791,
         0.1172372 ],
        ...,
        [0.08807629, 0.11803708, 0.21995674, ..., 1.        , 0.09234543,
         0.0474764 ],
        [0.09913997, 0.18797499, 0.40881791, ..., 0.09234543, 1.        ,
         0.04715232],
        [0.07081448, 0.06557388, 0.1172372 , ..., 0.0474764 , 0.04715232,
         1.        ]])

<2713x2713 sparse matrix of type '<class 'numpy.float64'>'
        with 7360369 stored elements in List of Lists format>
```

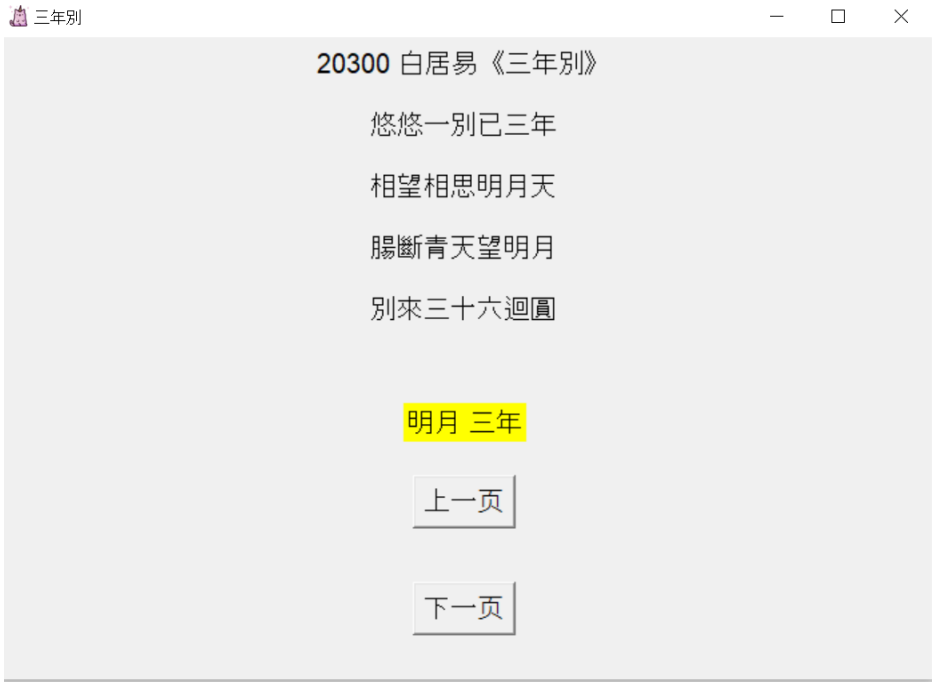
之后再直接通过计算词向量的 cosine 距离（利用 scipy 的库可以很方便的直接得出），就能得到词和词之间的相似程度了（距离越近，相似度越高），为了直观让相似度和值同向，所以采用了 1-cosine distance 进行处理。

```
rel = sparse.lil_matrix(1-pairwise_distances(sparse_M, metric="cosine"))
```

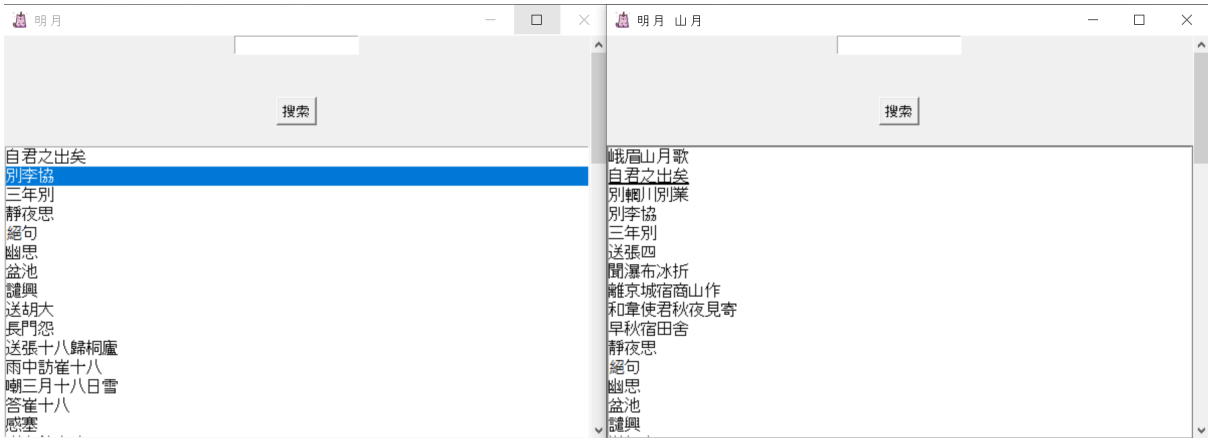
关键词前后对比

仅过滤作者（李白）

作者+关键词（白居易 酒 春天）



关键词（明月、山月）检索出来的其中一首诗歌



只检索明月，和复合检索明月、山月的对比

加分项

1. 设计合理的评价方案并据此设定**近义词的合理阈值方案**

   通过关键词，把其所在的那列的相似度排列，从最高的开始遍历，经实验，阈值首先排除99.5%的字（可通过计算相关矩阵除了 0 和 1 以外的值的 0.995 分位数得到），效果较好

```
for i in lst: #选最接近的字
    if i[1] < 0.68739376*f: #选出前0.05%匹配的字（当f=100%）
        break
```

   其中 f 为可以在界面手动动态调整的值，也可直接选择前 x 个，效果其实也相当。

2. 设计**合理的评价方案**并据此调参

   最终要选出的值的评价方案，不仅要考虑和近义词和关键词的相似度，还得考虑含有该近义词在其诗歌内的重要性。因此可先通过最基本的 TF-IDF 法衡量每个词在其诗歌的地位

	tf-idf																		
Poem_id	4371	4373	4394	4403	4408	4417	4418	4429	4431	4432	...	38901	38902	38905	38907	38910	39040	39092	39200
word																			
—	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
——	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一事	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一人	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一作	0.131584	0.0	0.0	0.0	0.0	0.0	0.0	0.103388	0.0	0.108557	...	0.083505	0.0	0.086846	0.086846	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
齋	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
齡	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龍	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龍門	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龜	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0

   两者合并，则根据关键词过滤出来的诗歌的价值可定义为$value = (similarity + importance) * ratio$，其中 similarity 为近义词与关键词的相似度，importance 为该词在诗歌里的重要性，ratio 为需要调整的参数，由近义词与关键词的相似顺位不断迭代相乘。

```
ratio = 1
for i in lst: #选最接近的字
    if i[1] < 0.68739376*f: #选出前0.05%匹配的字（当f=100%）
        break
    ratio *= 0.78
```

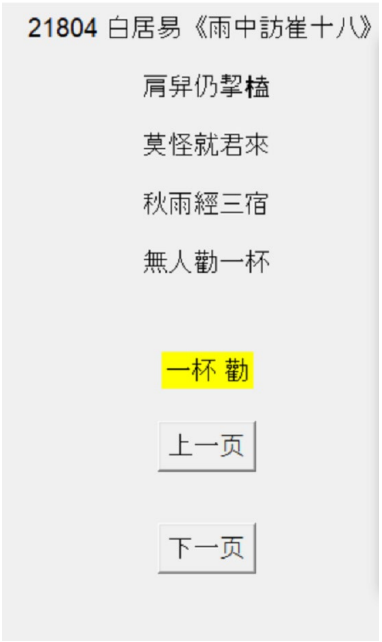
对于第一顺位（如明月/山月），ratio = 1，对于顺位为 2 的词（如“夜”），ratio=1*0.7=0.7，最后过滤选择 value 值最大的 50 首诗歌展示。

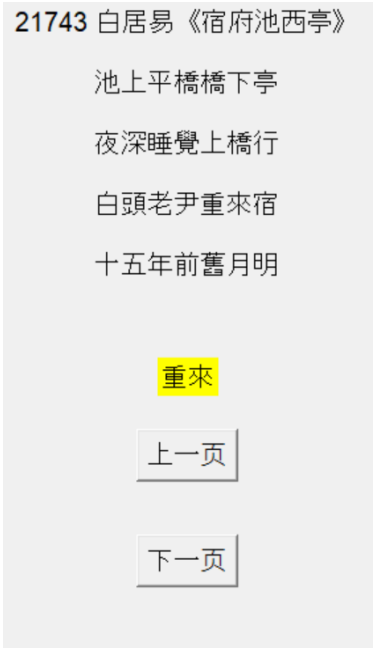
其他尝试：

不采用上下文，直接使用此表，以词是否同时出现在诗歌里进行相似度判断（cosine 距离），可以从另一个角度得到词的相似性（同理可得诗歌间的相似性）

	tf-idf																		
Poem_id	4371	4373	4394	4403	4408	4417	4418	4429	4431	4432	...	38901	38902	38905	38907	38910	39040	39092	39200
word																			
一	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一一	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一事	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一人	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
一作	0.131584	0.0	0.0	0.0	0.0	0.0	0.0	0.103388	0.0	0.108557	...	0.083505	0.0	0.086846	0.086846	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
齋	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
齡	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龍	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龍門	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0
龜	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0

调整一下其他参数，如对白居易 酒 春天会得到以下分别。





左图为基于词是否经常出现在同一首诗歌，右图为基于上下文的相似度