

A search engine for classical Chinese poems

Xiwei Wu & Ning Yang & Hui Ma

Abstract: This is a group-based project for Web Search & Mining. In this project, we implemented a poetry crawler that crawled more than 10,000 poems from 中华诗词网 from pre-Qin to modern times, then implemented a search engine that can support a variety of different search requirements, and finally provided a web front-end that can be easily used.

Keywords: Scrapy, Chinese poems, search engine

1 Project Requirements

In this project, we are required to crawl classical Chinese poems online and building a simple search engine. Detailed requirements are as follows:

1. Data Crawling and Preprocessing.

You need to crawl classical Chinese poems from websites. After that, you have to do some necessary data preprocessing steps to transform the raw data into structure records, with each having the following attributes: title, author, dynasty, content, and optionally other information if available: translation, annotation, appreciation, background, etc.

2. Search Engine. Based in the poems, you need to implement a simple search engine supporting the following functionalities.

a) **Boolean Search:** Users provide search keys and operations between keys. The system needs to return all the relevant poems (Either title or content meets the requirement of the query). The query language must include operations such as AND, OR, NOT.

b) **Zone-specific Search:** Allow users to specify the attribute to filter/search.

For enumerable attributes like author and dynasty, the results should only contain those exactly match the query.

For other attributes like title and content, a Boolean or ranked search can be performed. You can use either designs of UI or query (e.g. author(李白), title(酒)) to support such specification.

- c) **Pinyin-based Tolerant(Fuzzy) Search:** Just like SOUNDEX introduced in class, we can use pinyin for Chinese phonetic tolerant search. This can be especially useful when the user enters a wrong word with the same sound, or the poem itself contains 通假字. Therefore, your search engine should support a special query language SOUND(x) which can retrieve poems with any word having the same pinyin as x. For example, we should be able to get " 最爱湖东行不足，绿杨阴里白沙堤。" with SOUND(荫).
- d) **Ranked Search:** Return the search results with certain order that may be favorable by the users. The factors to consider may include: semantic relevance, fame of the author and the poem itself, etc. You can use any ranking method and any available information only to achieve this. We will prepare several test cases and score the system according to our search experience.

3. Nice Web Interface for demo and Project Report.

Outline. In Section 2, we will introduce our crawler based on Scrapy and SQL. In Section 3, we will introduce our design for search engine. In Section 4, we will introduce our design for web front-end. In Section 5, we will give some of our thoughts on the project implementation process.

2 Data Crawling

We use Scrapy for crawling data and SQL for storing data. Here we use 中华诗词网 as our poem source. Our crawler starts with the pages belonging to different dynasties. By analyzing the web code we get the relevant information.

```

1  def start_requests(self):
2      dynastys = {
3          "先秦": "https://www.shi-ci.com/dynasty/72057594037927936",
4          "汉代": "https://www.shi-ci.com/dynasty/144115188075855872",
5          "三国两晋": "https://www.shi-ci.com/dynasty/216172782113783808",
6          "南北朝": "https://www.shi-ci.com/dynasty/288230376151711744",
7          "隋代": "https://www.shi-ci.com/dynasty/360287970189639680",
8          "唐代": "https://www.shi-ci.com/dynasty/432345564227567616",
9          "宋代": "https://www.shi-ci.com/dynasty/504403158265495552",
10         "元代": "https://www.shi-ci.com/dynasty/576460752303423488",
11         "明代": "https://www.shi-ci.com/dynasty/648518346341351424",
12         "清代": "https://www.shi-ci.com/dynasty/720575940379279360",
13         "近现代": "https://www.shi-ci.com/dynasty/792633534417207296",
14     }
15     for k,url in dynastys.items():
16         yield Request(url,meta={"item":k},callback=self.parse_dynasty)

```

For each poem, we collect dynasty, poet name, poem name, contents, poet description and the associated page url.

```
1 class PoemItem(scrapy.Item):
2     dynasty = scrapy.Field()
3     poet_name = scrapy.Field()
4     poet_url = scrapy.Field()
5     poem_name = scrapy.Field()
6     poem_url = scrapy.Field()
7     contents = scrapy.Field()
8     poet_desc = scrapy.Field()
9     crawl_url = scrapy.Field()
```

Based on these data, we created the table via mysql, and the relevant steps can be referred to the poem/README.md file. Finally, we output the data in SQL format for later use. You can see our dataset as poem/data/poem.sql.

Due to the problem of website data, we need to process the poems obtained by the crawlers, such as the unification of Chinese and English commas, the addition of missing content in the body, and the appearance of ellipses due to the excessive length of titles. Here we would like to mention the ellipsis in the title, we found that among the more than 50,000 ancient poems we collected, there are more than 1,200 poems with ellipsis in the title, most of them are Tang poems. And we found that the titles of these poems existed in most of the websites in the form of containing ellipses, so we also dealt with only some of the poems for which the full names could be found.

3 Search Engine

We use Python to build the entire search engine, and a series of data structures, such as DataFrame are used to build our entire search engine.

3.1 Overall Design

The entire search process is as follows: first, based on the SQL database, we built several dictionaries and stored the entire SQL database into a DataFrame `df_all`, along with the rank information we decided. After a series of processes, our engine initialization is complete and enter the query phase.

After asking user to get the query type and query content, we process the content according to the query type, and then make the corresponding query within `df_all`, return the results for output, and ask the user whether to continue the query. We will describe how to implement these steps in the following sections.

All the codes are in '`process.py`' and '`run.py`'.

3.2 Data Process

First, after reading into the database, we stored it in the DataFrame `df_all` for easy processing, and the attributes are as previously mentioned : dynasty, poet name, poem name, contents, poet

description. Also, for subsequent zone-specific queries, we store two dictionaries of poem name and contents.

Then, for Boolean search, we construct a term dictionary. And consider the complexity of the grammar of Chinese ancient poetry, we decide to think a single word as a term. According to this idea, we construct the term dictionary `dict_word`. Based on the dictionary, we process and obtain a 0-1 vector for each ancient poem and construct the term-poem incidence matrix, and the matrix is `matrix_binary_all`. In each vector, 1 means the word appears in the poem corresponding to this position and 0 means not.

Finally, we appended two sets of ranking values to each ancient poem according to the ranking we thought about and added them to `df_all`. The entire data processing process is completed.

3.3 Boolean Search

The user input a string include search keys and operations, for each search key, we convert it to a 0-1 vector based on the dict we got before. For multiple single words inside the search key, we get all the corresponding vectors and then do **AND** on them. Then we check each result we get to make sure that the search key is actually present in the result, not scattered throughout the poem.

Then for each search key, we perform the corresponding **AND**, **OR**, **NOT** operation, and get the final result vector. And, unlike the documentation requirements, we have included dynasties and authors in the search

And finally, all search methods, including this one, will end up with a vector, and then the `vec_to_outlist` and `outlist_to_out` functions will be called to output the corresponding poems. This step is not repeated in the following search.

Here are three examples of searches.

```

输入查询类型, 0: binary; 1: zone; 2 : fuzzy; 3: ranked; 4: zone+ranked
0
请输入查询
白居易 AND 琵琶行
  Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
12853      2559      12853      12853      唐代      白居易      琵琶行      浔阳江头夜送客, \n枫叶荻花秋瑟瑟。 \n主人下马客在船, \n举酒欲饮无管弦。 \n醉不成欢惨将...
共搜到诗的数目:
1

```

图 1: Boolean Search example 1: search with AND. We search '白居易 AND 琵琶行', and return 1 result, which is correct.

```

输入查询类型, 0: binary; 1: zone; 2 : fuzzy; 3: ranked; 4: zone+ranked
0
请输入查询
床前明月光 OR 汉皇重色
  Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
12104      12104      12104      唐代      白居易      长恨歌      汉皇重色思倾国, \n御宇多年求不得。 \n杨家有女初长成, \n养在深闺人未识。 \n天生丽质难自...
  Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
22038      22038      22038      唐代      李白      静夜思      床前明月光, \n疑是地上霜。 \n举头望明月, \n低头思故乡。      855      30
共搜到诗的数目:
2

```

图 2: Boolean Search example 2: search with OR. We search '床前明月光 OR 汉皇重色', and return 1 result, which is correct.

```

输入查询类型, 0: binary; 1: zone; 2 : fuzzy; 3: ranked; 4: zone+ranked
0
请输入查询
张若虚
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
29325      29325      29325      唐代      张若虚      春江花月夜      春江潮水连海平, \n海上明月共潮生。 \n滟滟随波千万里, \n何处春江无月明。 \n江流宛转绕芳...
2
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
29328      29328      29328      唐代      张若虚      代答闺梦还      关塞年华早, \n楼台别望违。 \n试衫著暖气, \n开镜见春晖。 \n燕入窥罗幕, \n蜂来上画衣。 ...
2
共搜到诗的数目:
2

```

```

输入查询类型, 0: binary; 1: zone; 2 : fuzzy; 3: ranked; 4: zone+ranked
0
请输入查询
张若虚 AND NOT 春江花月夜
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
29328      29328      29328      唐代      张若虚      代答闺梦还      关塞年华早, \n楼台别望违。 \n试衫著暖气, \n开镜见春晖。 \n燕入窥罗幕, \n蜂来上画衣。 ...
2
共搜到诗的数目:
1

```

图 3: Boolean Search example 3: search with NOT. We search '张若虚', and return 2 result; if we search '张若虚 NOT 春江花月夜', and we will get 1 result, the specified item is excluded from the search results.

3.4 Zone-specific Search

The dynasty, author, title and content are entered in order, or keep empty. For enumerable attributes like author and dynasty, the returned vector only contain those exactly match the query, where we use the query function of DataFrame,

```

1 list = df_all.loc[df_all.dynasty_name == dynasty]
2 list = df_all.id[df_all.poet_name == author]

```

And we use Boolean search on title and content. At last, we do AND on the four returned vectors to get the final out, empty set as all-1 vector.

The example is shown in fig?? and fig??.

```

输入查询类型, 0: binary; 1: zone; 2 : fuzzy; 3: ranked; 4: zone+ranked
1
请输入朝代, 或者输入0保持空
唐代
请输入诗人名字, 或者输入0保持空
江城子
请输入诗题标题, 或者输入0保持空
江城子
请输入诗词内容, 或者输入0保持空
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
4266      4266      4266      唐代      韦庄      江城子      鬓鬟狼籍黛眉长, \n出兰房, \n刚睡醒。 \n角声呜咽, \n庭前斗草催花。 \n露冷月残人未起, \n...
337
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
17765      17765      17765      唐代      牛峤      江城子      晓飞起郡城东, \n碧江空, \n半滩风。 \n越王宫殿, \n绣叶藕花中。 \n帘卷水楼鱼浪起, \n...
12
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
18022      18022      18022      唐代      欧阳炯      江城子      晚日金陵岸草平, \n落霞明, \n水无情。 \n六代繁华, \n暗逐逝波声。 \n空有姑苏台上月, \n...
23
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
30749      30749      30749      唐代      张泌      江城子      记得千外小中庭, \n雨初晴, \n晓莺啼。 \n飞絮落花, \n时节近清明。 \n睡起卷帘无一事, \n...
29
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
33080      33080      33080      唐代      尹鹖      江城子      裙拖碧玉, \n步飘香, \n织成束素长。 \n露云光, \n拂面犹疑, \n玉纤凝夜。 \n宝柱秦筝弹...
12
Unnamed: 0      id dynasty_name poet_name poem_name      contents desc weight_poet weight_poem
34272      34272      34272      唐代      和凝      江城子      斗转星移玉漏频, \n已三更, \n对梧桐。 \n历历花间, \n似有马蹄声。 \n含蕊整衣开绣户, \n...
22
共搜到诗的数目:
6

```

图 4: Zone-specific Search example 1: We search '江城子' as title and '唐代' as dynasty, and return 6 results, all meet the requirements.

3.5 Pinyin-based Tolerant Search

To handle the fuzzy search based on pinyin, we created a table of Chinese characters to pinyin correspondence, which can be found in poem/data/pinyin.txt. For every word of the target to be

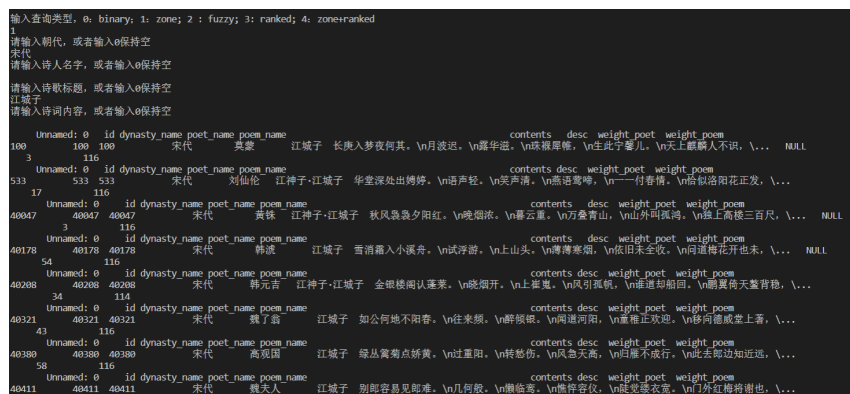


图 5: Zone-specific Search example 2: We search '江城子' as title and '宋代' as dynasty, and we can see that there are 103 results, and all meet the requirements.

searched, we will do single word substitution according to the pinyin, and do a Boolean search, then connect each different returned vector with **OR**.

Finally, we use **AND** to link all the single word vectors and output.

Such a search has several advantages and several disadvantages, the advantages of such search method is very powerful, as long as the pin-yin is right, it will definitely return the correct result, but there are corresponding cost. The number of results returned may be large, and the search time may be longer, there is no guarantee that the results the user wants will be at the top of the list. It may be possible to optimize further with a fuzzy search rank, but the search will take longer. Taking all things into account, we decided to keep this search the way it is now.

The example is shown in fig??.



图 6: Fuzzy Search example: We search '绿杨阴里白沙低', with correct '堤' is changed to the homophonic '低', and we can see that expected correct result is returned.

3.6 Ranked Search

First, the search result is based on Boolean search, but here we no longer support logical operations. We have thought and discussed, and have come up with some ranking bases. One caveat is that poetry ordering is very subjective and varies from person, so we do not guarantee that our ordering will be the best in all cases. Here are some thoughts:

Intuitively, for the same content, it appears at the beginning or at the end, we usually prefer

the former;

For the same content, if it appears in the poet, the title, the dynasty and the content of the poem, we want the poet is the first appears in the results, the title second, and the dynasty and the content.

For the fame of the poet, since there is no necessarily correct measure method, we used a more intuitive measure: the more poems this poet had, the more likely he was famous. For those poets who are very famous but have few works now, this may not be the right standard, but for most poets, this is the right standard.

And, for users, they always prefer short content to lengthy content, so we will put shorter poems in front.

Based on these idea, we measured each poem the number of poems retained by the author and the length of the poem, and add them to `df_all`. This process is very time-consuming(about 2-3 minutes), so we store the `df_all` as a csv so that we don't have to wait too long after one initialization.

Before the final output, we use `query_seq` to do these rank, first we sort the poems according to content's length, then we sort the poems in reverse order according to the the number of poems retained by the author, at last we sort the poems according to location where the query appears.

The more the author's work and the shorter the poem, the earlier the query is matched, the more this result will be in front.

For the other queries, since the query is composite, we only sorted the poem length and the number of poems, and the function is `query_seq_noquery`.

By the way, because of the presence of a large number of anonymous poets(“无名氏”), in order to ensure that they do not unduly affect the ranking of normal poets we manually weighted their number of poems to 1.

The example is shown in ??, we use same key to get different result return order.

4 Web Interface

5 Thoughts

During the implementation of this project, we tried many different websites, but due to the continuous development of crawlers and anti-crawlers, many websites have adopted the sweep copy or login to block crawlers. For general considerations, we have chosen this site to crawl. And Scrapy's sophisticated crawler template saves us a lot of time, we only need to do simple processing through the web code to get the data we need.

While doing pre-processing of the poem names and body, we found some poems with names containing Chinese characters not supported by utf8, so we decisively discarded them. Therefore, we can see that many poems contain vacant parts like * or "□", but of course, this is also partly due to the missing words in the process of circulation. Also, we found that the body of the poems

请输入查询类型: 0: binary; 1: zone; 2: fuzzy; 3: ranked; 4: zone+ranked

请输入查询

李白	Unnamed: 0	id	dynasty_name	poet_name	poem_name	contents	desc	weight_poet	weight_poem
1232	95	1232	1232	唐代	苏颋	送李白墓	夜郎归未老, \n醉死此江边, \n醉醒皆家礼, \n诗殊乐府篇, \n游魂应到蜀, \n小喝岂堪贫, ...		
2163	740	2163	2163	唐代	还人卷	李白李贺建机杆, \n敢在人间不知处, \n向君收在芙蓉江, \n斗蛟人织秋道, \n金忆礼礼文离, ...			
2251	740	2251	2251	唐代	读李白集	端云涛, \n引巨鳌, \n吸括造化空牢干, \n冥心入海海神稀, \n游龙不取为班主, \n人问物象, ...			
4177	337	4177	4177	唐代	布庄	过当涂县	客过当涂县, \n停车访旧游, \n谢公山若翠, \n李白酒无馐, \n石花空发, \n马江水自流, ...		
4241	38	4241	4241	唐代	布庄	洋亭驿小楼桃	当年此树正花开, \n匹马与仙郎载来, \n李白已亡工部死, \n何人堪伴玉山颓, ...		337
4245	38	4245	4245	唐代	布庄	焦崖阁	李白曾歌蜀道难, \n长闻白日上青天, \n今朝经过焦崖阁, \n始信星何在马前, ...		337
4605	322	4605	4605	唐代	杨愈	醉留东野	昔年因读李白杜陵诗, \n长恨二人不相从, \n尔曹与东野生并世, \n如何复随二子踪, \n东野不得, ...		
6028	464	6028	6028	唐代	陆龟蒙	停免院旧游	陵阳佳地昔曾游, \n谢眺青山李白楼, \n唯有日斜溪上思, \n道逢猿鸟影苍苍, ...		464

(a) 1

请输入查询类型: 0: binary; 1: zone; 2: fuzzy; 3: ranked; 4: zone+ranked

请输入搜索, 我们会对结果进行排序

李白	Unnamed: 0	id	dynasty_name	poet_name	poem_name	contents	desc	weight_poet	weight_poem
21933	21933	21933	唐代	李白	别东林寺僧	东林送客处, \n月出白猿啼, \n吴兴庐山远, \n僧向顶溪归, ...		855	30
21935	21935	21937	唐代	李白	初出金门, 寻王侍御不遇, 咏壁上鹤	高羽群金距, \n低鸣咤锦衣, \n能言犹见弄, \n还向陇西飞, ...			
30									
21966	21966	21968	唐代	李白	独坐敬亭山	众鸟高飞尽, \n孤云独去闲, \n相看两不厌, \n只有敬亭山, ...		855	30
21973	21973	21973	唐代	李白	玉阶怨	玉阶生白露, \n夜久侵罗袜, \n却下水晶帘, \n犹望望秋月, ...		855	30
21978	21978	21980	唐代	李白	秋浦歌·其十四	炉火照天地, \n红星乱紫烟, \n赧郎明月夜, \n歌曲动寒川, ...		855	30
21981	21981	21983	唐代	李白	秋浦歌·其十六	秋浦田舍翁, \n抱瓮灌中园, \n望子来白田, \n望子来白田, ...		855	30
21982	21982	21984	唐代	李白	秋浦歌·其十五	白发三千丈, \n缘愁似个长, \n不知明镜里, \n何处得秋霜, ...		855	30
21983	21983	21985	唐代	李白	秋浦歌·其十三	渌水净素月, \n月明白鹭飞, \n宿鸟惊寒夜, \n江上落残晖, ...		855	30
21985	21985	21987	唐代	李白	秋浦歌·其十二	水如一匹练, \n此地即平天, \n临风可举月, \n望花上渡船, ...		855	30
21986	21986	21988	唐代	李白	秋浦歌·其十七	试问一水间, \n谁道千里隔, \n举头忽见云间月, \n低头忽见水中月, ...		855	30

(b) 2

图 7: Rank Search example: we search '李白' in Boolean Search and in Rank Search, and we can find that, in Boolean Search, The first result is not a poem written by 李白, while for Rank Search, we can get it in the first result.

on this site contained an error message like "以上作品共五首", but we didn't deal with them all because there were too many of them.

As for search engine, we can have more improvements for fuzzy search, such as improving the pinyin database to match only the common passwords or misspellings in ancient poems to reduce the search space; on the other hand, we can also convert the final result into pinyin and combine it with the searched pinyin to sort the results. And for rank search, we can manually add a weight to those authors with few works but a strong reputation to get their results further up the list, for example, 张若虚 and 《春江花月夜》.

As for web, xxxxxxxxxx.

You can see our codes and dataset on the Github Crawl-for-poems.