

# Assignment 5

CS834-F16: Introduction to Information Retrieval

Fall 2016

Erika Siregar

CS Department - Old Dominion University

December 17, 2016

### Question 10.3

Compute five iterations of HITS (see Algorithm 3) and PageRank (see Figure 4.11) on the graph in Figure 10.3. Discuss how the PageRank scores compare to the hub and authority scores produced by HITS.

#### Answer

Figure 1 shows the directed graph from the textbook [1] on which we will calculate the scores of HITS and PageRank. Computing HITS (authorities and hubs) and PageRank scores are pretty easy since we can just utilize the Link Analysis procedure that is provided by python library 'networkx' [2].

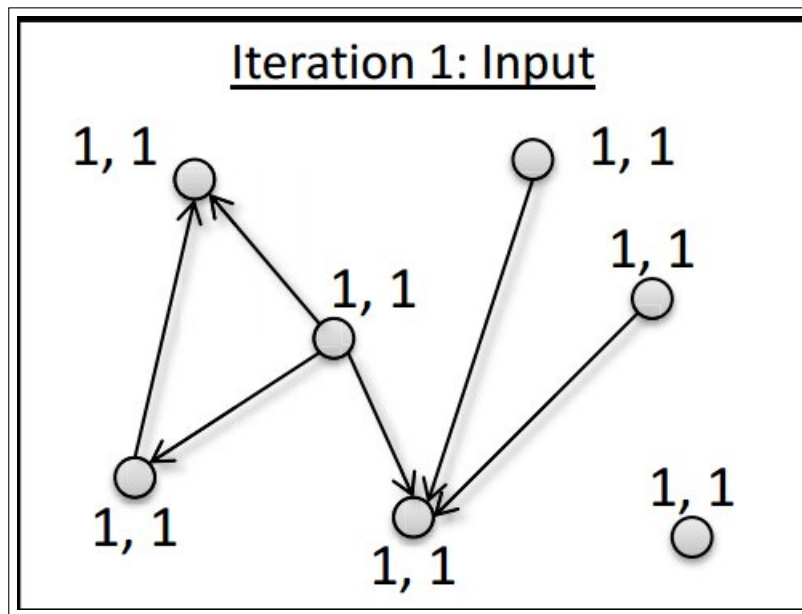


Figure 1: Figure 10.3 from the textbook [1]

Figure 2 shows the scores of HITS (authorities and hubs) and PageRank, which are obtained by running the code in listing 1. We only need to set the number of iterations.

```

erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assig
erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieva
l/assignments/a5/code_report$ PS1='\u:\W\$ '
erikaris:code_report$ PS1='\u@\h: '
erikaris@erikaris-Inspiron: python 10_3.py
HITS Algorithm (5 iterations)
=====
Hubs values = {1: 0.19870751068593528, 2: 0.19870751068593528, 3: 0.258854060655404,
4: 0.1789639731325056, 5: 0.0823834724201099, 6: 0.0823834724201099, 7: 0.0}
Authorities values = {1: 0.20153417015341699, 2: 0.20153417015341699, 3: 0.2523245002
3245, 4: 0.18816829381682937, 5: 0.07821943282194328, 6: 0.07821943282194328, 7: 0.0}

Pagerank Algorithm (5 iterations)
=====
Pagerank values = {1: 0.15415210590313183, 2: 0.15415210590313183, 3: 0.2151836555953
412, 4: 0.2730700543561289, 5: 0.08952435340504106, 6: 0.08952435340504106, 7: 0.0243
93371432183873}
erikaris@erikaris-Inspiron: █

```

Figure 2: HITS and Pagerank for Figure 10.3 with 5 Iterations

To make the analysis and comparison easier, I transformed the output into a neat table format as can be seen on table 1. From table 1, we can see that, generally, the authorities values are linearly proportional to those of PageRank. After 5 iterations, node 3 gets the highest score for ‘authorities’ and the second highest score for ‘PageRank’. Nodes 1 and 2 get lower ‘authorities’ score than that of node 3, but higher ‘authorities’ score compare to nodes 5 and 6. The same thing can also be concluded by comparing the PageRank scores for those five nodes (1, 2, 3, 5, and 6). The strange thing happens on node 4, where its ‘authorities’ score is lower than node 3, but its ‘PageRank’ score is higher than node 3. This anomaly takes place probably because we only do 5 iterations. Maybe, if we continue iterating until the values converge into certain number, this anomaly will not happen.

Node	Score		
	Hubs	Authorities	PageRank
1	0.198707510685935	0.201534170153416	0.154152105903131
2	0.198707510685935	0.201534170153416	0.154152105903131
3	0.258854060655404	0.252324500232450	0.215183655595341
4	0.178963973132505	0.188168293816829	0.273070054356128
5	0.082383472420110	0.078219432821943	0.089524353405041
6	0.082383472420110	0.078219432821943	0.089524353405041
7	0.000000000000000	0.000000000000000	0.024393371432184

Table 1: HITS and Pagerank for Figure 10.3 with 5 Iterations

```

1 #!/usr/bin/python
2
3 import networkx as nx
4

```

```

5 def hits(G, iter=100, nstart=None, normalized=True):
6     if type(G) == nx.MultiGraph or type(G) == nx.MultiDiGraph:
7         raise Exception("hits() not defined for graphs with multiedges.")
8     if len(G) == 0:
9         return {}, {}
10    # choose fixed starting vector if not given
11    if nstart is None:
12        h=dict.fromkeys(G, 1.0/G.number_of_nodes())
13    else:
14        h=nstart
15    # normalize starting vector
16    s=1.0/sum(h.values())
17    for k in h:
18        h[k]*=s
19    i=0
20    while True: # power iteration: make up to max_iter iterations
21        if i >= iter: break
22
23        hlast=h
24        h=dict.fromkeys(hlast.keys(), 0)
25        a=dict.fromkeys(hlast.keys(), 0)
26        # this "matrix multiply" looks odd because it is
27        # doing a left multiply  $a^T = hlast^T * G$ 
28        for n in h:
29            for nbr in G[n]:
30                a[nbr] += hlast[n] * G[n][nbr].get('weight', 1)
31        # now multiply  $h = Ga$ 
32        for n in h:
33            for nbr in G[n]:
34                h[n] += a[nbr] * G[n][nbr].get('weight', 1)
35        # normalize vector
36        s=1.0/max(h.values())
37        for n in h: h[n]*=s
38        # normalize vector
39        s=1.0/max(a.values())
40        for n in a: a[n]*=s
41
42        i+=1
43        if normalized:
44            s = 1.0/sum(a.values())
45            for n in a:
46                a[n] *= s
47            s = 1.0/sum(h.values())
48            for n in h:
49                h[n] *= s
50        return h, a
51
52    def pagerank(G, alpha=0.85, personalization=None,
53        iter=100, nstart=None, weight='weight',
54        dangling=None):
55        if len(G) == 0:
56            return {}
57
58        if not G.is_directed():
59            D = G.to_directed()
60        else:
61            D = G
62
63        # Create a copy in (right) stochastic form

```

```

64 W = nx.stochastic_graph(D, weight=weight)
65 N = W.number_of_nodes()
66
67 # Choose fixed starting vector if not given
68 if nstart is None:
69     x = dict.fromkeys(W, 1.0 / N)
70 else:
71     # Normalized nstart vector
72     s = float(sum(nstart.values()))
73     x = dict((k, v / s) for k, v in nstart.items())
74
75 if personalization is None:
76     # Assign uniform personalization vector if not given
77     p = dict.fromkeys(W, 1.0 / N)
78 else:
79     missing = set(G) - set(personalization)
80     if missing:
81         raise nx.NetworkXError('Personalization dictionary '
82                                 'must have a value for every node. '
83                                 'Missing nodes %s' % missing)
84     s = float(sum(personalization.values()))
85     p = dict((k, v / s) for k, v in personalization.items())
86
87 if dangling is None:
88     # Use personalization vector if dangling vector not specified
89     dangling_weights = p
90 else:
91     missing = set(G) - set(dangling)
92     if missing:
93         raise nx.NetworkXError('Dangling node dictionary '
94                                 'must have a value for every node. '
95                                 'Missing nodes %s' % missing)
96     s = float(sum(dangling.values()))
97     dangling_weights = dict((k, v/s) for k, v in dangling.items())
98     dangling_nodes = [n for n in W if W.out_degree(n, weight=weight) == 0.0]
99
100 # power iteration: make up to max_iter iterations
101 for _ in range(iter):
102     xlast = x
103     x = dict.fromkeys(xlast.keys(), 0)
104     danglesum = alpha * sum(xlast[n] for n in dangling_nodes)
105     for n in x:
106         # this matrix multiply looks odd because it is
107         # doing a left multiply  $x^T = xlast^T W$ 
108         for nbr in W[n]:
109             x[nbr] += alpha * xlast[n] * W[n][nbr][weight]
110     x[n] += danglesum * dangling_weights[n] + (1.0 - alpha) * p[n]
111
112 return x
113
114 if __name__ == '__main__':
115     iter = 5
116     G = nx.Graph()
117
118     # Add 7 nodes
119     G.add_nodes_from(range(1,8))
120
121     # Add 6 edges
122     G.add_edges_from([(1,2), (3,1), (3,2), (3,4), (5,4), (6,4)])

```

```

123
124 # Compute hubs and authorities normalized values using hits
125 h, a = hits(G, iter=iter)
126
127 print 'HITS Algorithm ({0} iterations)'.format(iter)
128 print '=====',
129 print 'Hubs values = {}'.format(h)
130 print 'Authorities values = {}'.format(a)
131 print ''
132
133 # Compute pagerank of each nodes
134 pr = pagerank(G, iter=iter)
135
136 print 'Pagerank Algorithm ({0} iterations)'.format(iter)
137 print '=====',
138 print 'Pagerank values = {}'.format(pr)

```

Listing 1: Computing HITS and PageRank

## Question 10.5

Find a community-based question answering site on the Web and ask two questions, one that is low-quality and one that is high-quality. Describe the answer quality of each question.

### Answer:

For this assignment, I asked 2 questions on 2 different community-based question answering site. For the low-quality question, I asked about ‘*What is the purpose of our life?*’<sup>1</sup> on Yahoo Answers <https://answers.yahoo.com/> as can be seen on figure 3. For the high-quality question, I asked the question ‘*BUILD-MAX-HEAP running time for array sorted in decreasing order*’ on Stackoverflow <sup>2</sup> as can be seen on figure 5.

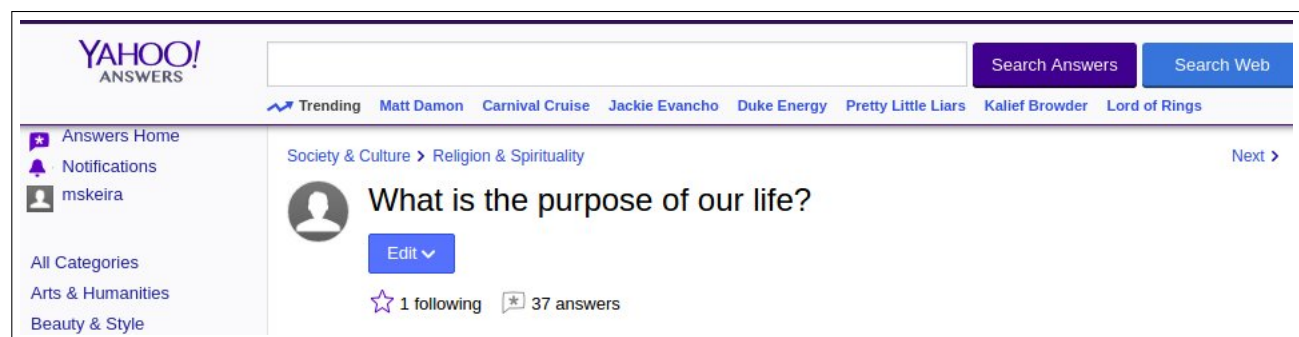


Figure 3: Low Quality Question I asked on Yahoo Answers

<sup>1</sup><https://answers.yahoo.com/question/index?qid=20161216122515AAPvTIw&page=4>

<sup>2</sup><http://stackoverflow.com/questions/39691923/build-max-heap-running-time-for-array-sorted-in-decreasing-order>




Figure 4: High Quality Question I asked on Stackoverflow

By the time I write this report, I got 37 answers for the low-quality question on Yahoo Answers. Some of the answers can be seen on figure 5. For the high-quality question on Stackoverflow, I only got 2 answers as can be seen on figure 6.

However, for the low-quality question, I also got low-quality answers. This is understandable because for a low-quality question, people tend to post anything that they have in their minds without being afraid of any risks. For example, when I asked 'What is the purpose of our life', I got answers like 'Who says that there's one?', 'To study the paintings of the great Masters, and understand that UFOs are real', and 'Drink beer and have a good time'.



Accordingly, despite the low number of answers, I got high-quality answers for the high-quality question. This is understandable because for this type of question, people will think twice (or maybe more) before submitting the answers. They should be able to provide not only answer, but also the explanation why the answer is correct. People will not take the risk of embarrassing themselves by saying something irrelevant. For a high-quality answer, not everyone has the ability to provide a justifiable answer and explanation. Hence, we got a lower number of answers for the high-quality question compare to the low-quality question.




To Live, Procreate and Die...  
Although there is no need to procreate...  
It is rather pleasurable to go through the motions.

~



Everard · 21 hours ago


[Award Best Answer](#)  0  0



To enjoy the ride and be happy 😊😊



Fox · 22 hours ago


[Award Best Answer](#)  3  0



To reproduce and keep the species alive, nothing more than that.



Julius · 22 hours ago


[Award Best Answer](#)  6  3



Drink beer and have a good time

spicy · 22 hours ago

[Award Best Answer](#)  6  3



According to Jesus' first Christians, it is for Christ to make Gods out of us.

<https://en.wikipedia.org/wiki/Divinizati...>

Publius · 15 hours ago



[Award Best Answer](#)  0  0

Figure 5: Answers for the low-quality question I asked on Yahoo Answers



▲

0

▼

✓

If you *know* that the array is already sorted in decreasing order, then there's no need to sort it. If you want it in ascending order, you can reverse the array in  $O(n)$  time.


If you don't know whether the array is already sorted, then it takes  $O(n)$  to determine if it's already reverse sorted.

The reason building a max heap from a reverse-sorted array is considered  $O(n)$  is that you have to start at item  $n/2$  and, working backwards, make sure that the element is not smaller than its children. It's considered  $O(n)$  even though there are only  $n/2$  checks, because the number of operations performed is proportional to the total number of items to be checked.

It's interesting to note, by the way, that you can build a max-heap from a reverse-sorted array faster than you can check the array to see if it's reverse sorted.

share edit

answered Sep 27 at 17:42



Jim Mischel

89.9k ● 9 ● 95 ● 194

add a comment

---

▲

0

▼

✓

You are right. It can of course be  $O(1)$ . When you know for sure that your list is sorted you can use it as your max heap.

The common implementation of a heap using array uses this behavior for its elements position:

```
childs[i] = 2i+1 and 2i+2
parent[i] = floor((i-1)/2)
```

This rule applies on a sorted array. (descending for max-heap, increasing for min-heap).


Please **note** that if you need to check first that the list is sorted it is of course still  $O(n)$ .

**EDIT: Heap Sort Complexity**  
Even though the array might be sorted and building the heap might actually take  $O(1)$ . Whenever you perform a Heap Sort you will still end up with  $O(n \log n)$ . As said in the comments, Heap Sort is performing  $n$  calls to `extract-max`. Each extraction operation takes  $O(\log n)$  - We end up with total time complexity of  $O(n \log n)$ . In case the array is not sorted we will get total time-complexity of  $O(n + n \log n)$  which is still  $O(n \log n)$ .

share edit

edited Sep 27 at 8:32

answered Sep 25 at 21:59



A. Sarid

2,721 ● 1 ● 9 ● 38

---

Thank you for your answer. But, why does the Heap Sort still have running time of  $O(n \lg n)$ , even though the BUILD-MAX-HEAP only takes time  $O(1)$ . Why does it not  $O(1 \lg n) = O(\lg n)$ ? – [mskeira](#) Sep 25 at 23:07

@mskeira because the time of performing heap sort is dominated by  $n$  calls to `extract-max`, each of which takes  $\log n$  time, even if the heap is initially fully sorted. There are some variant implementations of heap sort which perform better on "nearly sorted" inputs. – [rici](#) Sep 26 at 0:54

@mskeira I've edited my answer accordingly eventhough it is a bit out of the scope of the question. If you find this answer correct, please consider accepting it by clicking on the V next to my answer. Glad to help. – [A. Sarid](#) Sep 26 at 6:33

Figure 6: Answers for the high-quality question I asked on Stackoverflow

## Question 10.6

Find two examples of document filtering systems on the Web. How do they build a profile for your information need? Is the system static or adaptive?

### Answer

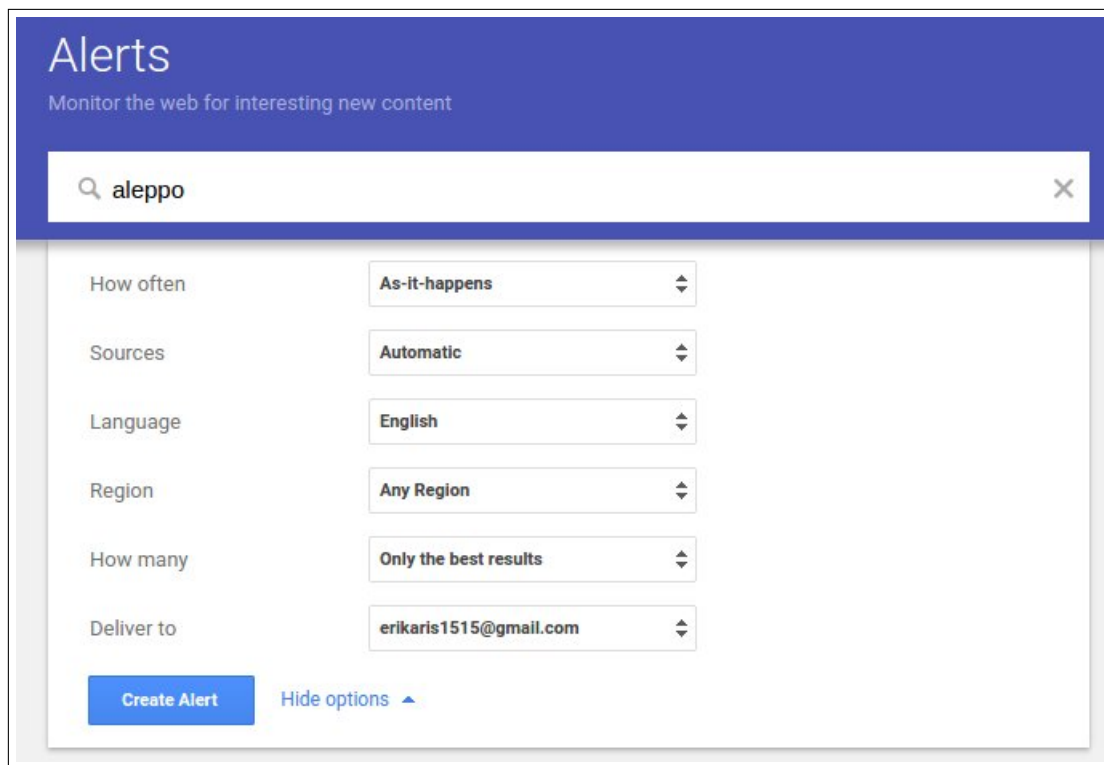
I found 2 examples of document filtering systems on the Web, which are:

1. Google Alerts (<https://www.google.com/alerts>). This is a content change detection and notification service, offered by the search engine company Google.
2. Twilert (<https://www.twilert.com/>). This is a tool to get realtime alerts anytime a certain keyword you are interested in are mentioned on Twitter.

Both Google Alerts and Twilert use the same method to build the profile: using input from the user. Figure 7 and 8 shows the creation of profile on Google Alerts and Twilert, respectively.

To build a profile on Google Alerts, a user needs to input several information:

1. The keyword that the user is interested in. This keyword will be the profile's name.
2. Sources: news, blogs, videos, etc
3. Language: English, Arabic, etc
4. Region: select a country's name
5. How many: best results, all results.
6. Deliver to: user's email address.



Alerts	
Monitor the web for interesting new content	
<input type="text" value="aleppo"/>	
How often	As-it-happens
Sources	Automatic
Language	English
Region	Any Region
How many	Only the best results
Deliver to	erikaris1515@gmail.com
<input type="button" value="Create Alert"/> <a href="#">Hide options</a>	

Figure 7: Creating a profile on Google Alerts

Similarly, to build a profile on Twilert, user also needs to input several information such as the keyword, the location of the tweets, and the type of the tweets (positive, negative, or question).

Create new Twilert

Refine Schedule

Refine your search ([help?](#))

Press tab or enter to add

aleppo x Enter your search

geocode: 29.2985278, 42.550960300000004, 2639.9593967541023km x

exclude:retweets x

Filter by word(s) Filter by user Filter by location Misc

10937 potential results

Next

Figure 8: Creating a profile on Twilert

The key to determine if a filtering system is static or adaptive is to check whether the profile is updateable or not. Based on this definition, I can conclude that both Google Alerts and Twilert are using adaptive filtering system since they provide a menu to update/edit profile. Figure 9 and 10 illustrate the profile update for Google Alerts and Twilert, respectively. In this case, I make the keyword more specific by changing 'aleppo' to 'aleppo evacuation'. For Twilert, I also specify the tweet location to 'Syria' (figure 11).

A screenshot of the Google Alerts settings interface. At the top, a search bar contains the text "aleppo evacuation" with a magnifying glass icon on the left and a close "X" icon on the right. Below the search bar, there are several settings, each with a label on the left and a dropdown menu on the right: "How often" is set to "At most once a day", "Sources" is set to "News", "Language" is set to "English", "Region" is set to "Syria", "How many" is set to "Only the best results", and "Deliver to" is set to "erikaris1515@gmail.com". At the bottom left, there is a blue button labeled "Update alert" which is circled in red. To its right is a link that says "Hide options" with a small upward-pointing triangle.

Figure 9: Update a profile on Google Alerts

A screenshot of the Twilert search settings page. The background is dark blue. At the top, the text "Edit Twilert" is centered and circled in red. Below it is a toggle switch with a blue circle on the left labeled "Refine" and a grey circle on the right labeled "Schedule". Underneath the toggle is the text "Refine your search (help?)". In the center, there is a white search box containing two grey tags: "aleppo evacuation" (circled in red) and "exclude:retweets". Below these tags is a grey box labeled "geocode:" (circled in red) containing the text "34.80207499999999, 38.996814999999997, 415.9886492544569 km". Below the search box is the text "Enter your search". At the bottom, there are four buttons: "Filter by word(s)", "Filter by user", "Filter by location", and "Misc", each with a downward arrow. Below the filters, on the left, is the text "151 potential results". On the right, there is a large blue button labeled "Next".

Figure 10: Update a profile on Twilert

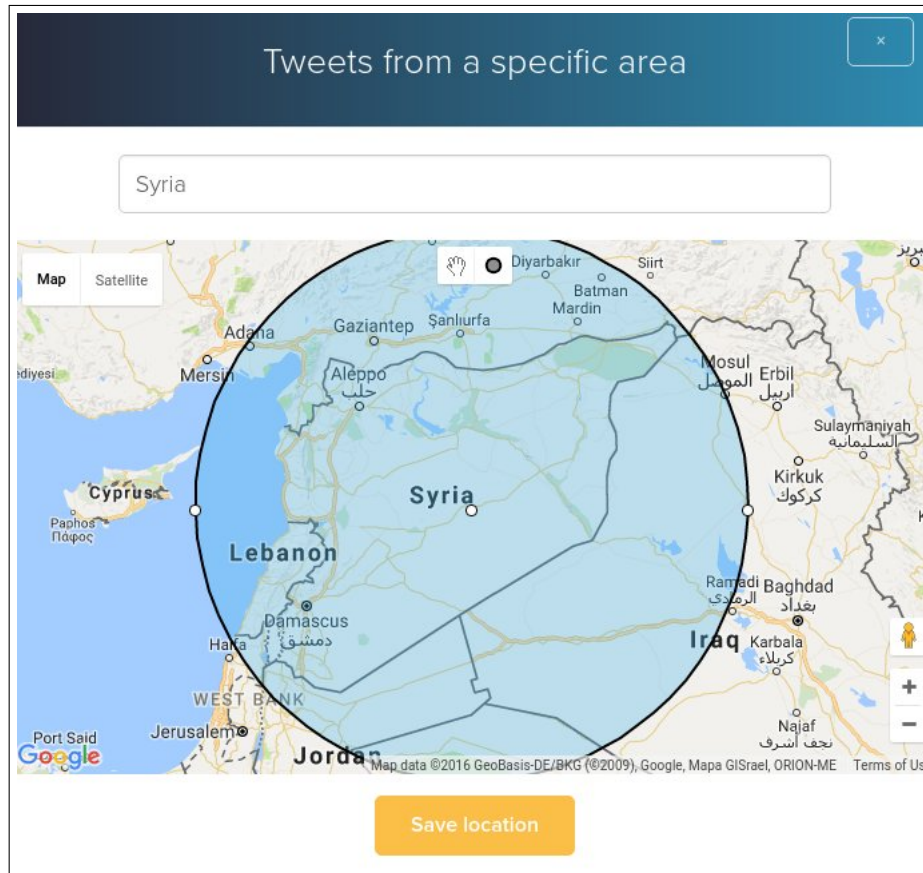


Figure 11: Update a profile on Twilert - update the area

Figure 12 and 13 show the filtering result from Google Alerts and Twilert, respectively.

Alert preview

There are no recent results for your search query. Below are existing results that match your search query.

NEWS

---

**Rebels surrender east **Aleppo**, **evacuations** begin**  
Syria Direct  
AMMAN: Rebel forces say they are prepared to leave east **Aleppo** on Thursday, as civilians and the wounded depart for the city's opposition-controlled ...

---

**'Aleppo will remain a mark of shame on the international community': 5 residents say goodbye**  
Syria Direct  
The **evacuation** of civilians and fighters from the 3km pocket of land still held by Syrian opposition forces in east **Aleppo** city began on Thursday.

Rebel forces down to final few districts in east **Aleppo** - Syria Direct  
Ceasefire in **Aleppo** as eastern half burns, 'bodies lie where they fell' - Syria Direct  
Ceasefire falls apart as artillery, airstrikes rock east **Aleppo** - Syria Direct  
[Full Coverage](#)

---

**Civilians increasingly cornered amid ongoing rebel losses in east **Aleppo**, 'stranglehold getting ...**  
Syria Direct  
Civilians increasingly cornered amid ongoing rebel losses in east **Aleppo**, ... calls for a mass civilian **evacuation** went unanswered by Russian and Syrian regime ... The regime's ongoing campaign to retake **Aleppo** relies heavily on ...

---

Figure 12: Filter result by Google Alert



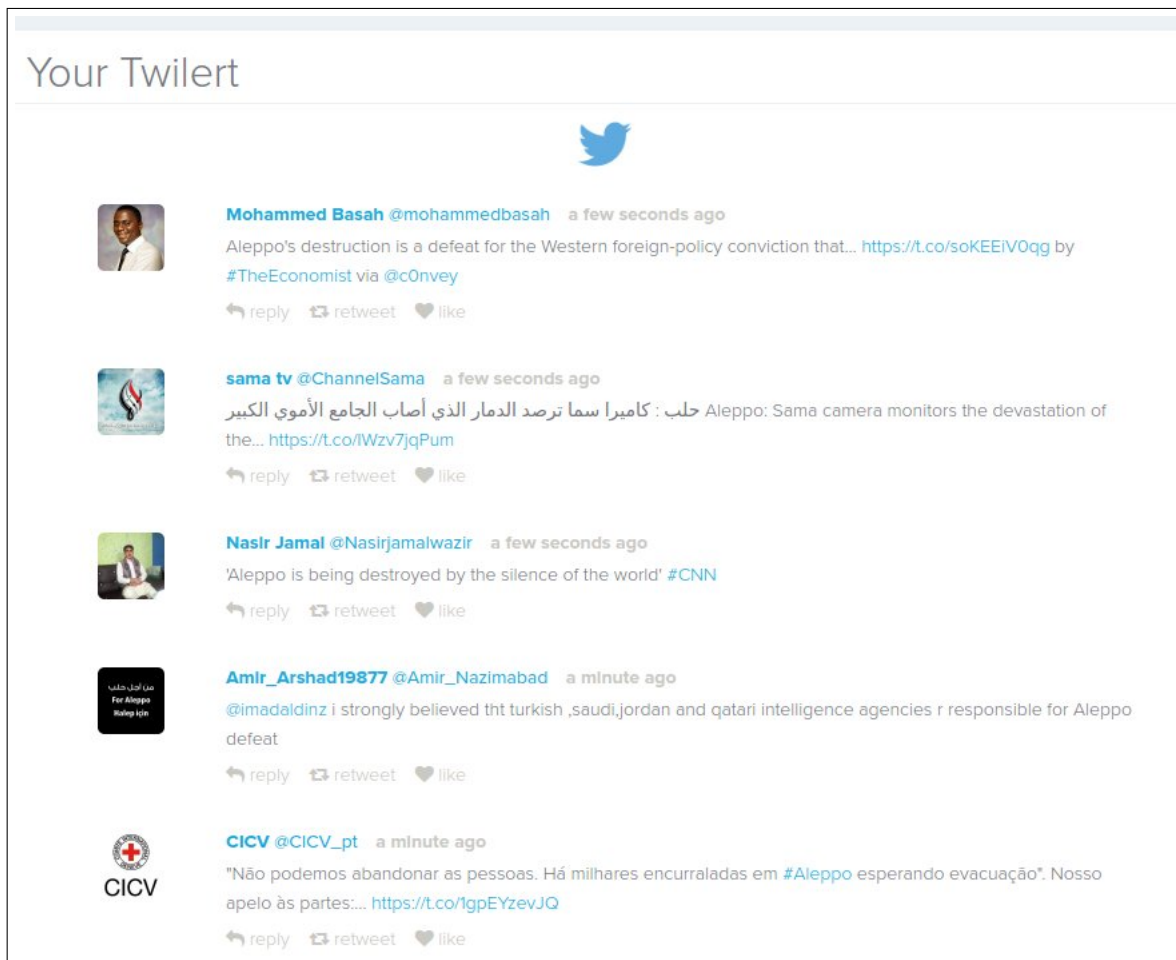


Figure 13: Filter result by Twilert

## Question 7.5

Implement a BM25 module for Galago. Show that it works and document it.

## Answer

Information about how to implement BM25 on Galago can be found on Lemur Project website [3]. First, we have to make the json query file. The format for the json query is available on [3] and it looks like the example on listing 2.

```

1  "queries" : [
2  {
3  "number" : "bm25-combine",
4  "text" : "#combine(#bm25(international) #bm25(organized) #bm25(crime))",
5  }

```

Listing 2: json query format for BM25

For this assignment, the json query that I use can be found on listing 3.

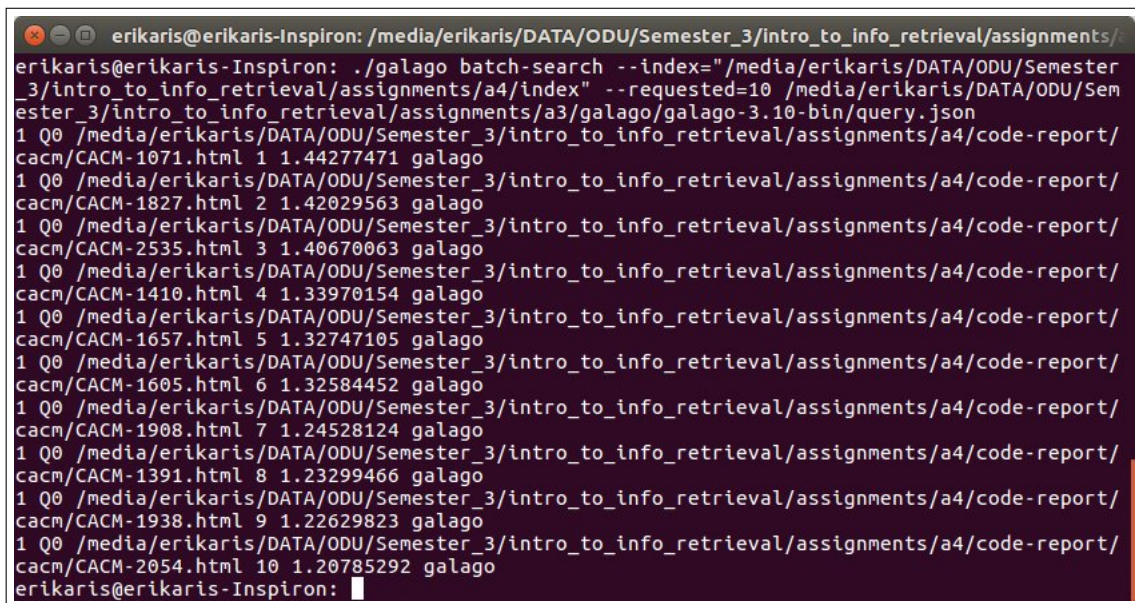
```

1  {
2  "query": [
3  {
4  "text": "#combine(#bm25(what) #bm25(articles) #bm25(exist) #bm25(which) #bm25(deal)
      #bm25(with) #bm25(tss) #bm25(time) #bm25(sharing) #bm25(system) #bm25(an) #bm25(
      operating) #bm25(system) #bm25(for) #bm25(ibm) #bm25(computers))",
5  "number": "1"
6  }
7  ]
8  }

```

Listing 3: json query format for BM25

To execute this query, use the command ‘./galago batch-search --index="[path to the index files]" --requested=10 [path to the json query]’. Output of this execution is shown on figure 14.



```

erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/
erikaris@erikaris-Inspiron: ./galago batch-search --index="/media/erikaris/DATA/ODU/Semester
_3/intro_to_info_retrieval/assignments/a4/index" --requested=10 /media/erikaris/DATA/ODU/Sem
ester_3/intro_to_info_retrieval/assignments/a3/galago/galago-3.10-bin/query.json
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1071.html 1 1.44277471 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1827.html 2 1.42029563 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-2535.html 3 1.40670063 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1410.html 4 1.33970154 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1657.html 5 1.32747105 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1605.html 6 1.32584452 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1908.html 7 1.24528124 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1391.html 8 1.23299466 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-1938.html 9 1.22629823 galago
1 Q0 /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a4/code-report/
cacm/CACM-2054.html 10 1.20785292 galago
erikaris@erikaris-Inspiron:

```

Figure 14: Output of BM25 implementation using Galago

## Question 7.8

Using the Galago implementation of query likelihood, study the impact of short queries and long queries on effectiveness. Do the parameter settings make a difference?

### Answer

According to [4], the formula for query likelihood is what can be seen on figure 15.



## Query Likelihood Model

- Unigram query likelihood

$$P(Q | \theta D) = \prod_{i=1}^n P(q_i | D)$$

$$P(q_i | D) = \frac{tf_{q_i, D}}{|D|} \quad \leftarrow \text{Maximum likelihood (ML) estimate, defined as: Tf of query term appearing in doc divided by document length}$$

Example:

*Q*: “computer virus,”

$p(\text{computer}|D) = 0.1, p(\text{virus}|D) = 0.05 \rightarrow p(Q|\theta D) = 0.1 * 0.05 = 0.005$

Figure 15: Query Likelihood Formula

This formula has 2 drawbacks:

1. Results in zero if a term is missing in document (figure 16)
2. Document may be relevant to query but the query term is absent from document

- If a term in a query does not occur in a document, the whole similarity measure becomes zero
- Example:
  - Q*: “gold silver truck”
  - D*<sub>1</sub>: “Shipment of gold damaged in a fire”
  - D*<sub>2</sub>: “Delivery of silver arrived in a silver truck”
  - D*<sub>3</sub>: “Shipment of gold arrived in a truck”
- Term *Silver* does not appear in *D*<sub>1</sub>. Similarly, *silver* does not appear in *D*<sub>3</sub> and *gold* does not appear in *D*<sub>2</sub>.
- This would result in Score=0 for all 3 documents.

$$p_{ml}(\text{silver} | \theta D_i) = \frac{tf(\text{silver}, D_i)}{|D_i|} = 0$$

Figure 16: Problem with missing document. Figure is taken from [4]

To overcome these issues, smoothing is needed. By default, Galago uses DirichletScorer for smoothing [5]. Figure 17 shows the snippet for DirichletScorer code that can be found in file ‘DirichletScoringIterator.java’ line 56 in Galago Project.

```
public double score(int count, int length) {
    double numerator = count + (mu * background);
    double denominator = length + mu;
    return Math.log(numerator / denominator);
}
```

Figure 17: Snippet for Dirichlet code in Galago Project

Using short query could impact the effectiveness since short query might contains noises (things that are not really relevant) in the result set. On the other hand, using a very long query could lower the likelihood value, especially if the long query contain a lower proportion of keywords. For example, the query 'search engines' may produce a better result with a web search engine than the query "“what are typical implementation techniques and data structures used in search engines”. Therefore, the best thing to do is to use an average-length query since it has enough keywords to produce a good result, but contain less noise compare to the short query. Parameter setting (query, query format, input file format, output file format, etc) will have insignificant effect on the query effectiveness. It normally will not affect the top-ranked pages of the search engine result.

---

## Question 10.8

Implement the nearest neighbor-based collaborative filtering algorithm. Using a publicly available collaborative filtering data set, compare the effectiveness, in terms of mean squared error, of the Euclidean distance and correlation similarity.

### Answer

For this assignment, I use the movielens dataset which is available at <http://grouplens.org/datasets/movielens/100k/>. I wrote a python program that utilize python libraries 'scipy' and 'sklearn'. The complete code can be found on listing 4.

Based on the output on figure 18 we can see that the MSE of Euclidean distance is smaller than the MSE of the correlation similarity (Pearson). Therefore, in this case, we can conclude that Euclidean distance is more effective compare to correlation similarity (Pearson).

```
erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval
erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a5/code_report$ python 10_8.py
Mean Squared Error of Euclidean: 1.427125
Mean Squared Error of Pearson: 2.166100
erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a5/code_report$
```

Figure 18: Output for question 10\_8

```
1
2 #!/usr/bin/python
3 from numpy import genfromtxt, array, random
4 from sklearn.linear_model import LinearRegression
5 from sklearn.metrics import mean_squared_error
6 from sklearn.neighbors import KNeighborsClassifier
7
8 if __name__ == '__main__':
9     # Read data ratings.csv from movielens
10    ratings = genfromtxt('ratings.csv', delimiter=',', skip_header=1, usecols=range(3))
11
12    # Split data to train and test data, with ratio train:test = 0.9:0.1
13    random.shuffle(ratings)
14    train, test = ratings[len(ratings)/10:, :], ratings[:len(ratings)/10, :]
15
16    # Train data using KNN, get 1st top neighbor
17    neigh = KNeighborsClassifier(n_neighbors=1)
18    # X = 2-dimensional array, y = target = 1st col of X
19    neigh.fit(train, train[:, 0])
20    # Get neighbors indices (2-dimensional array)
21    distances, neighbors_indices = neigh.kneighbors(test)
22    # Get ratings from indices
23    euclidean_neighbors = array([train[ns[0]] for ns in neighbors_indices])
24
25    # Train data using Pearson
26    lr = LinearRegression()
27    # X = 2-dimensional array, y = target = 1st col of X
28    lr.fit(train, train[:, 0])
29    # Get neighbors index (1-dimensional array)
30    neighbors_idx = lr.predict(test)
31    # Get ratings from indices
```

```

32 pearson_neighbors = array([train[int(idx)] for idx in neighbors_idx])
33
34 # Get MSE of predicted ratings, for both euclidean and pearson
35 y_true = []
36 for user, movie, rating in test:
37     y_true.append(rating)
38
39 y_pred_euc = []
40 for user, movie, rating in euclidean_neighbors:
41     y_pred_euc.append(rating)
42
43 y_pred_pear = []
44 for user, movie, rating in pearson_neighbors:
45     y_pred_pear.append(rating)
46
47 print "Mean Squared Error of Euclidean: %f" % mean_squared_error(y_true, y_pred_euc)
48 print "Mean Squared Error of Pearson: %f" % mean_squared_error(y_true, y_pred_pear)

```

Listing 4: Code for nearest neighbor-based collaborative filtering algorithm

---

## Question 10.9

Both the clustering and nearest neighbor-based collaborative filtering algorithms described in this chapter make predictions based on user/user similarity. Formulate both algorithms in terms of item/item similarity. How can the distance between two items be measured?

### Answer

For this assignment, I still use the movielens dataset from <http://grouplens.org/datasets/movielens/100k/>. I also utilize the python code ‘recommendation.py’ provided in the book ‘Programming Collective Intelligence’ [6]. The code can be seen on listing 5. The algorithm to compute the item similarity is similar to that of user similarity. We only need to ‘flip’ our point of view, from measuring the similarity between users to measuring similarity between items (see listing 5, line 14). We still use the same distance metrics: Correlation similarity and Euclidean distance. Figure 19 shows the snippet of the output for question 10.9.



```

erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a5/code_report
erikaris@erikaris-Inspiron: /media/erikaris/DATA/ODU/Semester_3/intro_to_info_retrieval/assignments/a5/code_report$ python 10_9.py
Using euclidean distance : Actual movie = To Cross the Rubicon (1991), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = To Cross the Rubicon (1991), Predicted movie = ♦ koldum klaka (Cold Fever) (1994)
Using euclidean distance : Actual movie = Birdcage, The (1996), Predicted movie = Aiqing wansui (1994)
Using pearson similarity : Actual movie = Birdcage, The (1996), Predicted movie = Maya Lin: A Strong Clear Vision (1994)
Using euclidean distance : Actual movie = B. Monkey (1998), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = B. Monkey (1998), Predicted movie = ♦ koldum klaka (Cold Fever) (1994)
Using euclidean distance : Actual movie = Little Big League (1994), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Little Big League (1994), Predicted movie = Killing Fields, The (1984)
Using euclidean distance : Actual movie = Chairman of the Board (1998), Predicted movie = 1-900 (1994)
Using pearson similarity : Actual movie = Chairman of the Board (1998), Predicted movie = Wings of the Dove, The (1997)
Using euclidean distance : Actual movie = Mortal Kombat (1995), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Mortal Kombat (1995), Predicted movie = Roommates (1995)
Using euclidean distance : Actual movie = Tin Men (1987), Predicted movie = 8 Heads in a Duffel Bag (1997)
Using pearson similarity : Actual movie = Tin Men (1987), Predicted movie = Pather Panchali (1955)
Using euclidean distance : Actual movie = Endless Summer 2, The (1994), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Endless Summer 2, The (1994), Predicted movie = Natural Born Killers (1994)
Using euclidean distance : Actual movie = Village of the Damned (1995), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Village of the Damned (1995), Predicted movie = Down Periscope (1996)
Using euclidean distance : Actual movie = Careful (1992), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Careful (1992), Predicted movie = ♦ koldum klaka (Cold Fever) (1994)
Using euclidean distance : Actual movie = Air Bud (1997), Predicted movie = 8 1/2 (1963)
Using pearson similarity : Actual movie = Air Bud (1997), Predicted movie = Murder in the First (1995)
Using euclidean distance : Actual movie = Johnny 100 Pesos (1993), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Johnny 100 Pesos (1993), Predicted movie = Willy Wonka and the Chocolate Factory (1971)
Using euclidean distance : Actual movie = Bye Bye, Love (1995), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Bye Bye, Love (1995), Predicted movie = Program, The (1993)
Using euclidean distance : Actual movie = Men With Guns (1997), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Men With Guns (1997), Predicted movie = Trainspotting (1996)
Using euclidean distance : Actual movie = Beautiful Thing (1996), Predicted movie = 'Til There Was You (1997)
Using pearson similarity : Actual movie = Beautiful Thing (1996), Predicted movie = To Wong Foo, Thanks for Everything! Julie Newmar (1995)
Using euclidean distance : Actual movie = 2 Days in the Valley (1996), Predicted movie = 8 Heads in a Duffel Bag (1997)

```

Figure 19: A snippet of the output for question 10.9

```

1
2 #!/usr/bin/python
3
4 import os
5 import pickle
6 import random
7
8 import recommendations as r
9
10 if __name__ == '__main__':
11     __, __, prefs = pickle.load(open('movielens.pkl'))
12

```

```

13 # To make item-based, transform prefs, so the format of prefs will be prefs[movie][
    user] = rating
14 prefs = r.transform_prefs(prefs)
15
16 # Split prefs into train and test prefs
17 movies = prefs.keys()
18 random.shuffle(movies)
19
20 movies_train, movies_test = movies[:int(0.9 * len(movies))], movies[int(0.1 * len(
    movies)):]
21 train = {m: prefs[m] for m in movies_train}
22 test = {m: prefs[m] for m in movies_test}
23
24 for movie in test:
25     sim_distances = []
26     sim_pearsons = []
27
28     for other_movie in train:
29         # Calculate distance using euclidean distance
30         sim_distances.append((r.sim_distance(prefs, movie, other_movie), other_movie))
31         # Calculate similarity using pearson
32         sim_pearsons.append((r.sim_pearson(prefs, movie, other_movie), other_movie))
33
34     # distance sort ascending
35     sim_distances.sort()
36     # similarity sort descending
37     sim_pearsons.sort(reverse=True)
38
39     # select 1st top of the list
40     sim_most_related_movie = sim_distances[0][1]
41     pear_most_related_movie = sim_pearsons[0][1]
42
43     # Compare
44     print 'Using euclidean distance : Actual movie = {}, Predicted movie = {}'.format(
        movie, sim_most_related_movie)
45     print 'Using pearson similarity : Actual movie = {}, Predicted movie = {}'.format(
        movie, pear_most_related_movie)

```

Listing 5: Code for question 10.9

---

## References

- [1] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.
- [2] NetworkX Developers. Networkx - Link Analysis. [https://networkx.github.io/documentation/networkx-1.9/reference/algorithms.link\\_analysis.html](https://networkx.github.io/documentation/networkx-1.9/reference/algorithms.link_analysis.html), 2016. [Online; accessed 14-December-2016].
- [3] David Fisher. The Lemur Project / Wiki / Galago Operators. <https://sourceforge.net/p/lemur/wiki/Galago%20Operators/>, 2016. [Online; accessed 16-December-2016].
- [4] Nazli Goharian. Language Model. <http://people.cs.georgetown.edu/~nazli/classes/ir-Slides/LanguageModel-13.pdf>, 2016. [Online; accessed 16-December-2016].

- [5] Galago Project. The Lemur Project / Discussion / Galago:About retrieval models. <https://sourceforge.net/p/lemur/discussion/galago/thread/10ff53c2/>, 2016. [Online; accessed 16-December-2016].
- [6] Toby Segaran. *Programming Collective Intelligence*. International series of monographs on physics. O'Reilly Media, 2007.