INFO 4300: HW II

Jason Marcell

October 9th, 2010

1 Introduction

The object of this assignment was to use singular value decomposition for a recommendation engine and a search engine.

2 A Small-scale Recommender System

2.1 Singular Values

```
1   26.885239
2   23.857636
3   21.676541
4    3.876193
5    2.600910
6    0.804787
7    0.663664
8    0.496925
9    0.000000
10   0.000000
```

Above are listed the singular values. I chose to keep the top three because there is a sharp drop off after those three.

2.2 Independent Parameters

In order to compute the number of independent parameters, we use this equation from slide 8 of Lecture 11's lecture notes:

$$r(M+N)r^2 \tag{1}$$

Given that we have a $20x10$ matrix and we are taking $r = 3$ singular values, we obtain $3(20+10) - 3^2 = 81$ independent parameters.

2.3 Recommendations to Existing Users

By comparing X_k to the original X, what are the three strongest recommendations you would make to the existing users? (i.e., examine largest values of $X_k - X$)

Here we have a listing of $X_k - X$. (I have scaled up by a factor of 100 so that larger values stand out more heavily in the listing):

1	2	−5	2	−3	12	1	−2	1	−3	1
2	−54	−12	−54	−32	−14	18	223	−64	−32	21
3	7	3	7	4	2	−2	−30	6	4	−2
4	−44	−27	−44	−29	−14	6	199	12	−29	2
5	2	−7	2	−4	18	1	−2	1	−4	1
6	21	8	21	13	7	−7	−90	17	13	−6
7	2	−7	2	−4	18	1	−2	1	−4	1
8	5	−25	5	−0	−1	17	3	−4	−0	−3
9	6	−34	6	−0	−2	23	4	−6	−0	−4
10	4	−15	4	−0	−1	−36	2	15	−0	36
11	5	−25	5	−0	−1	17	3	−4	−0	−3
12	28	11	28	17	9	−9	−120	22	17	−8
13	3	−13	3	−3	12	6	−1	−1	−3	−0
14	14	5	14	8	4	−5	−60	11	8	−4
15	1	−2	1	−1	6	0	−1	0	−1	0
16	1	−10	1	16	−65	3	3	3	16	3
17	−47	232	−47	2	5	−43	−23	−29	2	−55
18	21	8	21	13	7	−7	−90	17	13	−6
19	10	−14	10	4	1	9	−28	3	4	−4
20	3	−10	3	−6	24	1	−3	2	−6	2

So, for each row, examining the top three values, we have recommendations for each user:

1. 5, 1, 2
2. 7, 10, 6
3. 1, 3, 8
4. 7, 8, 6
5. 5, 1, 3
6. 1, 3, 8
7. 5, 1, 3

8. 5, 1, 3
9. 10, 8, 1
10. 6, 1, 3
11. 1, 3, 8
12. 5, 6, 1
13. 1, 3, 4
14. 5, 1, 3
15. 4, 9, 6
16. 5, 4, 9
17. 1, 3, 8
18. 1, 3, 6
19. 5, 1, 3

2.4 Recommendations to New Users

Now we consider four new users. The last user is a user we know nothing about. We model this by giving that user uniform preference.

```
newuser1 = [6 0 0 0 0 0 0 0 0 0]
newuser2 = [0 0 0 0 0 2 0 0 0 4]
newuser3 = [0 3 0 0 0 0 0 3 0 0]
newuser4 = [1 1 1 1 1 1 1 1 1 1]
```

For each user, we perform the following steps

- Right multiply the vector by V to transform to feature space.
- Zero out the entries corresponding to the truncation of Σ.
- Transform the result back to user coordinates by right multiplying by V^T
- Compare with the initial user vector.

```
 1  User 1
 2  6.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
 3  2.47   1.01   2.47  -0.17  -0.10  -0.16   1.19  -0.14  -0.17  -0.25
 4
 5  User 2
 6  0.00   0.00   0.00   0.00   0.00   2.00   0.00   0.00   0.00   4.00
 7 -0.22   1.08  -0.22  -0.02   0.07   2.43  -0.13   1.22  -0.02   2.44
 8
 9  User 3
10  0.00   3.00   0.00   0.00   0.00   0.00   0.00   3.00   0.00   0.00
11  0.43   0.75   0.43  -0.00   0.04   1.16   0.21   0.57  -0.00   1.15
12
13  User 4
14  1.00   1.00   1.00   1.00   1.00   1.00   1.00   1.00   1.00   1.00
15  1.03   1.00   1.03   1.13   0.60   1.13   0.88   0.58   1.13   1.10
```

In the above listing we see each user vector and their corresponding result after performing the above-stated steps. We arrive at the following recommendations by observing the top three results for each user (disregarding songs they have already rated).

- User 1: 3, 7, 2
- User 2: 10, 8, 2
- User 3: 6, 10, 1
- User 4: 4, 6, 9

3 A Search Engine That Indexes Full Text Using Latent Semantic Indexing

For this part, I created a term-document matrix consisting of the tf.idf scores of each word. I then performed singular value decomposition, keeping the top two singular values and recomputed by matrix as C_k. Then, for the given query, I create a query vector. The query vector is a very long, sparse vector that is mostly zero's, but contains one's in the slots where the words form the query correspond to the entries. In order to score each document, I take the inner product of the query vector with the column vector corresponding to each document. I then sort the documents by score and return the top three (if available). If no documents scored greater than zero then nothing is returned.

NOTE! In order to get this part to work *reasonably* I had to vastly decrease the corpus. I limited my corpus to only 6 files and I truncated the length of the files in half. As it is, the indexing takes a while, but with the full 30-document corpus it is unreasonably slow.

4 Instructions for Running

4.1 Part 1

The results from part one were obtained from the unit tests located in `svd-test.rb`. You can run the unit test like this `ruby svd-test.rb --name test_homework_part_1`.

4.2 Part 2

Just run `ruby main.rb`. You'll see a prompt asking you to wait while it indexes. Then when it says "Ready!" you can type your query just like in the previous assignment. The program ends when you type "ZZZ"

```
$ ruby main.rb
Indexing. Please wait...
.
.
.
.
.
.
Ready!
vector
development
./test/file06.txt 0.810614700199872
... competitive fund by: Establishing a fast-track competitive
    grant process through the EDA 4 To build l ...
./test/file05.txt 0.620749063959116
... ver said Solutions to health issues here on Earth have the
    potential to benefit space explorers of th ...
```

5 References

- Formulas were used as per definition from the class website: `http://www.infosci.cornell.edu/Courses/info4300/2010fa/index.html`
- I collaborated with Karan Kurani for this project.