A Needle in a Data Haystack - HW3

Introduction to Data Science

Submitter: Roy Zohar 209896174

Problem 1: (Stemming)

(a) The identity rule SS → SS is important for words that end in SS such as CARESS, since these words are already stemmed. If SS → SS wasn't included, the CARESS would turn to CARES (different word entirely). This is because SS is a suffix we don't want to get rid of, but S → could destroy.

(b) PONIES → PONI

TIES → TI

CIRCUS → CIRCU

(c) I would add the rule SSES → SS. Trivially this rule turns CARESSES to CARESS. Note that the rule SES → S, or ES → , would turn horses for example into hors.

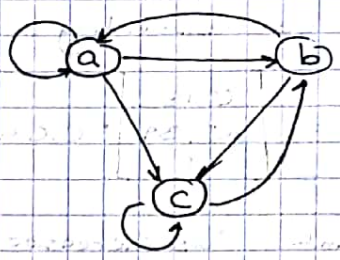
(d) Note that information, informal and informant all have different meanings, but are all stemmed down to "inform".

(e) Mouse → mouse, mice → mice

die → die, dice → dice

elf → elf, elves → elv.

Problem 2: (Page Rank)



(a) Assume no teleports. Therefore our equations are:

$$r_a = \frac{r_a}{3} + \frac{r_b}{2}$$

$$r_b = \frac{r_a}{3} + \frac{r_c}{2}$$

$$r_c = \frac{r_c}{2} + \frac{r_a}{3} + \frac{r_b}{2}$$

Then the matrix will be:

$$\begin{bmatrix} r_a \\ r_b \\ r_c \end{bmatrix} = \begin{bmatrix} 1/3 & 1/2 & 0 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 \end{bmatrix} \times \begin{bmatrix} r_a \\ r_b \\ r_c \end{bmatrix}$$

The eigenvectors of matrix M, that corresponds to the eigenvalue 1 is: $(1/2 \quad 2/3 \quad 1)$. After normalization, we get:

$$\begin{bmatrix} r_a \\ r_b \\ r_c \end{bmatrix} = \begin{bmatrix} 1/2 \cdot 6/13 \\ 2/3 \cdot 6/13 \\ 1 \cdot 6/13 \end{bmatrix} = \begin{bmatrix} 0.23 \\ 0.31 \\ 0.46 \end{bmatrix}$$

(b) Our new matrix M will be as follows:

$$M = 0.7 \cdot \begin{bmatrix} 1/3 & 1/2 & 0 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 \end{bmatrix} + 0.3 \cdot \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

Again, the corresponding eigenvector, normalized will be:

$$\begin{bmatrix} r_a \\ r_b \\ r_c \end{bmatrix} = \begin{bmatrix} 13/20 \\ 20/27 \\ 1 \end{bmatrix} \cdot 0.42 = \begin{bmatrix} 0.27 \\ 0.31 \\ 0.42 \end{bmatrix}$$

(c) $M = 0.7 \cdot \begin{bmatrix} 1/3 & 1/2 & 0 \\ 1/3 & 0 & 1/2 \\ 1/3 & 1/2 & 1/2 \end{bmatrix} + 0.3 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

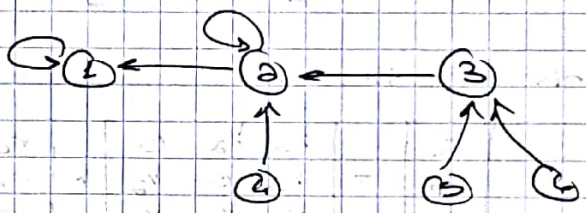
Once again, the corresponding eigenvector, normalized:

$$\begin{bmatrix} r_a \\ r_b \\ r_c \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.39 \\ 1 \end{bmatrix} \cdot 0.64 = \begin{bmatrix} 0.11 \\ 0.25 \\ 0.64 \end{bmatrix}$$

Note that c's rank has increased substantially due to it being the only node in the teleport set.

(d) After (e)

(e) Let G_3 be the following graph:



where the nodes marked 1, 2, 3 are the 3 relevant nodes. $S = \{1, 2, 3\}$

Note that for $\beta = 0.1$, the eigenvectors corresponding to $\lambda = 1$ is:

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{bmatrix} = \begin{bmatrix} 0.002 \\ 0.03 \\ 0.06 \\ 0.3 \\ 0.3 \\ 0.3 \end{bmatrix}$$ max rank is for node 3

---

* The M matrix for this graph is:

$$M = \beta \cdot \begin{pmatrix} 1 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} + (1-\beta) \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 & 1/3 & 1/3 & 1/3 \end{pmatrix}$$

And I define the teleport set as $\{4, 5, 6\}$

Now for $\beta = 1/2$, we get:

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{bmatrix} = \begin{bmatrix} 0.11 \\ 0.22 \\ 0.16 \\ 0.16 \\ 0.16 \\ 0.16 \end{bmatrix}$$

max rank is for node 2.

Finally, for $\beta = 0.9$, we get

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{bmatrix} = \begin{bmatrix} 0.96 \\ 0.02 \\ 0.0066 \\ 0.003 \\ 0.003 \\ 0.003 \end{bmatrix}$$

max rank is for node 1.

$\Longrightarrow$ $S = \{1, 2, 3\}$ satisfies the requirement and therefore our C_3 is valid.

Intuition: As the chance for teleport rises, there is a greater chance to land on node 3 since it has a lot of nodes pointing to it. On the other hand, Node 1 is more central when there are no teleports. Node 2 is a balance of nodes 1 and 3.

תרגיל בית 3 - מחט בערימת דאטה - תרגיל בונוס

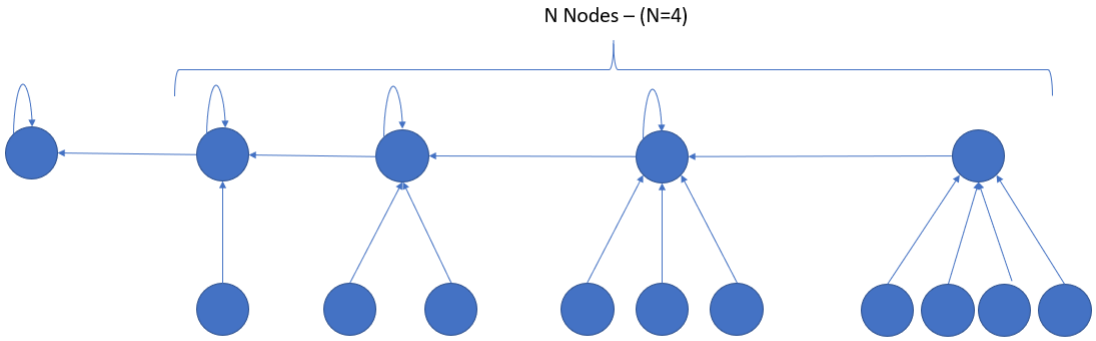
רועי זהר 209896174 ויואב תמיר 322291519

7 בינואר 2019

הערה: לא הצלחנו לפתור את תרגיל הבונוס בצורה עצמאית ולכן פתרנו אותו יחדיו. אנו מקווים שעדיין נוכל לקבל ניקוד בונוס על התרגיל.

פתרון:

נבחר את הגרף G_N הבא:

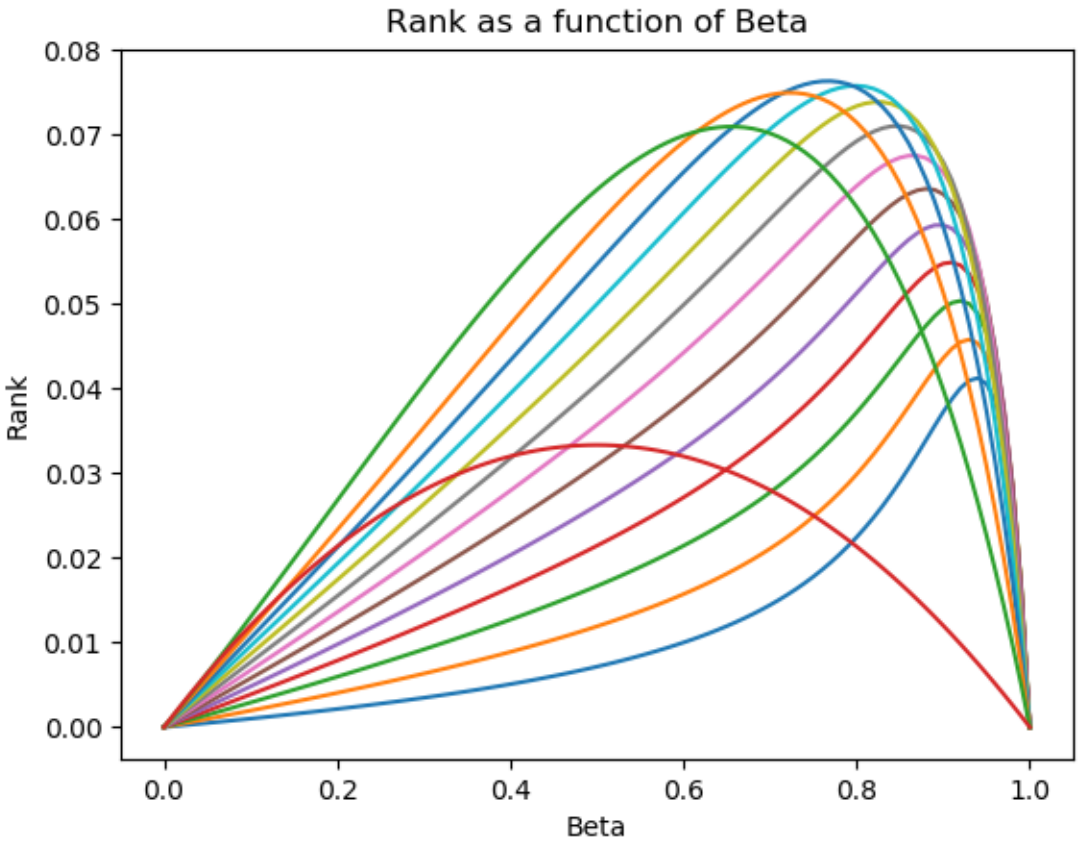


כאשר ה$Teleport\,Set$ שלנו הוא כל ה$nodes$ התחתונים (יש $\binom{N}{2}$ קודקודים כאלה). כמובן שניתן להמשיך גרף זה, ולבנות גרף כזה לכל N.

אינטואיציה: ככל שנקטין את בטא, יש יותר סיכוי ל$Teleport$ ולכן נפגע בסבירות גבוהה יותר ב$nodes$ מה$Teleport\,Set$, ובכך נגדיל את החשיבות של ה$nodes$ הימניים, וכנל להפך.

לא הצלחנו לחשב בצורה מתטית את החשיבות כפונקציה של בטא, ולכן רשמנו קוד שמוצא לנו בדיוק את החשיבות של כל $node$.

כאשר הרצנו את הקוד שלנו על $N = 14$, קיבלנו את הגרף הבא:



ואכן כשבדקנו, לכל $node$ הייתה בטא שעבורו הוא המקסימלי. הקוד שרשמנו מצורף:

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  EXAMPLE_A3 = np.array([
6      [1,1/2,0,0,0,0],
7      [0,1/2,1,1,0,0],
8      [0,0,0,0,1,1],
9      [0,0,0,0,0,0],
10     [0,0,0,0,0,0],
11     [0,0,0,0,0,0]
12 ])
13
14 EXAMPLE_B3 = np.array([
15     [0,0,0,0,0,0],
16     [0,0,0,0,0,0],
17     [0,0,0,0,0,0],
18     [1/3,1/3,1/3,1/3,1/3,1/3],
19     [1/3,1/3,1/3,1/3,1/3,1/3],
20     [1/3,1/3,1/3,1/3,1/3,1/3],
21 ])
22
23
24 def create_A(N):
25     A = []
26     TELEPORTS = N * (N - 1) // 2
27     TOTAL_NODES = N + N * (N - 1) // 2
28
29     for row_index in range(N):
30         row = [0] * TOTAL_NODES
31         if row index == 0:
```

```
31          if row_index == 0:
32              row[0] = 1
33              row[1] = 1/2
34          elif row_index == N - 2:
35              row[row_index] = 1/2
36              row[row_index + 1] = 1
37          elif row_index == N - 1:
38              pass
39          else:
40              row[row_index] = 1/2
41              row[row_index + 1] = 1/2
42
43          start = N + (row_index - 1) * row_index // 2
44          end = start + row_index
45          for index in range(start, end):
46              row[index] = 1
47          A.append(row)
48
49      for row_index in range(N, TOTAL_NODES):
50          row = [0] * TOTAL_NODES
51          A.append(row)
52
53      return np.array(A)
54
55  def create_B(N):
56      B = []
57
58      TELEPORTS = N * (N - 1) // 2
59      TOTAL_NODES = N + N * (N - 1) // 2
60
61      for row index in range(N):
```

```
61      for row_index in range(N):
62          row = [0] * TOTAL_NODES
63          B.append(row)
64
65      for row_index in range(TELEPORTS):
66          row = [1 / TELEPORTS] * TOTAL_NODES
67          B.append(row)
68
69      return np.array(B)
70
71  N = 15
72  RESOLUTION = 100
73
74  A = create_A(N)
75  B = create_B(N)
76
77  all_ranks = []
78  top_rank = []
79  for BETA in range(RESOLUTION + 1):
80      BETA /= RESOLUTION
81      sum = BETA * A + (1 - BETA) * B
82
83      eig_values, eig_vectors = np.linalg.eig(sum)
84      eig_pairs = [
85          ((eig_values[i]), eig_vectors[:,i])
86          for i in range(len(eig_values))
87      ]
88      eig_pairs.sort(key=lambda x: x[0], reverse=True)
89
90      ranks = eig_pairs[0][1]
91      ranks = ranks / np.sum(ranks)
```

```
91          ranks = ranks / np.sum(ranks)
92
93          top_rank.append(max(enumerate(ranks[1:N]), key=lambda x: x[1]
94
95          all_ranks.append(ranks[1:N])
96
97      print("Found Nodes: ", list(set(top_rank)))
98
99      plt.plot(np.array(list(range(RESOLUTION + 1))) / RESOLUTION, all_
100     plt.title("Rank as a function of Beta")
101     plt.xlabel("Beta")
102     plt.ylabel("Rank")
103     plt.show()
104
```