Data Mining Technology for Business and Society

Homework # 2

1536242 - Paolo Tamagnini

May 2016

1 Tables

Tables for each graph containing a simple statistic on the used datasets:

BUP	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	105	85	105	85	105	78	105	83	105	76
Num Edges	352	89	353	88	353	88	353	88	353	88
Number of pairs of nodes to analyze for solving Link-Prediction problem.	5108		5107		5107		5107		5107	

UAL	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	320	196	316	194	321	198	320	209	318	197
Num Edges	1700	413	1701	407	1701	412	1701	409	1701	409
Number of pairs of nodes to analyze for solving Link-Prediction problem.	49340		48069		49659		49339		48702	

INF	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	407	341	406	346	409	344	409	354	406	342
Num Edges	2212	549	2212	547	2212	552	2212	552	2212	549
Number of pairs of nodes to analyze for solving Link-Prediction problem.	80409		80003		81224		81224		80003	

SMG	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	992	656	999	655	992	642	996	652	997	647
Num Edges	3932	945	3933	951	3933	947	3933	950	3933	954
Number of pairs of nodes to analyze for solving Link-Prediction problem.	487604		494568		487603		491577		492573	

EML	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	1087	744	1102	754	1099	732	1092	745	1103	771
Num Edges	4360	1039	4361	1051	4361	1045	4361	1043	4361	1053
Number of pairs of nodes to analyze for solving Link-Prediction problem.	585881		602290		598990		591325		603392	

YST	Trainin Graph_0	Test Graph_0	Trainin Graph_1	Test Graph_1	Trainin Graph_2	Test Graph_2	Trainin Graph_3	Test Graph_3	Trainin Graph_4	Test Graph_4
Num Nodes	2112	1006	2121	1007	2124	1019	2110	999	2127	1017
Num Edges	5316	1147	5317	1154	5317	1157	5317	1139	5317	1159
Number of pairs of nodes to analyze for solving Link-Prediction problem.	2223900		2242943		2249309		2219678		2255684	

2 List of the used methods for LinkPrediction

- **Jaccard**

  Given two nodes, u and v compute the two sets of neighbors, U and V.Then perform the union and the intersection and compute their sizes. The Jaccard score will be the following:

$$score(u, v) = \frac{|U \cap V|}{|U \cup V|}$$

- **AdamicAdar**

  Compute the set of common neighbours z_i between u and v, then compute the amount of neighbours of each of them, $|Z_i|$. The score will be then:

$$score(u, v) = \sum_{z_i \in U \cap V} \frac{1}{\log |Z_i|}$$

  This will allow us to weight less pairs of node with extremely popular neighbours in common, meaning neighbours with large degree.

- **Preferential Attachment**

  This method is based on the idea that "rich will get richer", meaning that a node with already many links will gain more links more likely than a node poor of links. The score is indeed computed as the product between the number of neighbours of u and v:

$$score(u, v) = |U| \cdot |V|$$

- **Number of Common Neighbours**

  Pretty straightforward, it is the following:

$$score(u, v) = |U \cap V|$$

- **Shortest Path Length**

  Given two nodes u and v, we compute the shortest path between them. Then we compute the length of such path as the number of nodes visited, the scorer will be higher for shorter paths and lower for longer ones. If there is no path, the pair should get the lowest score possible: the total number of nodes in the graph.

$$score(u, v) = -length(shortest\ path(u, v))$$

3 Plots

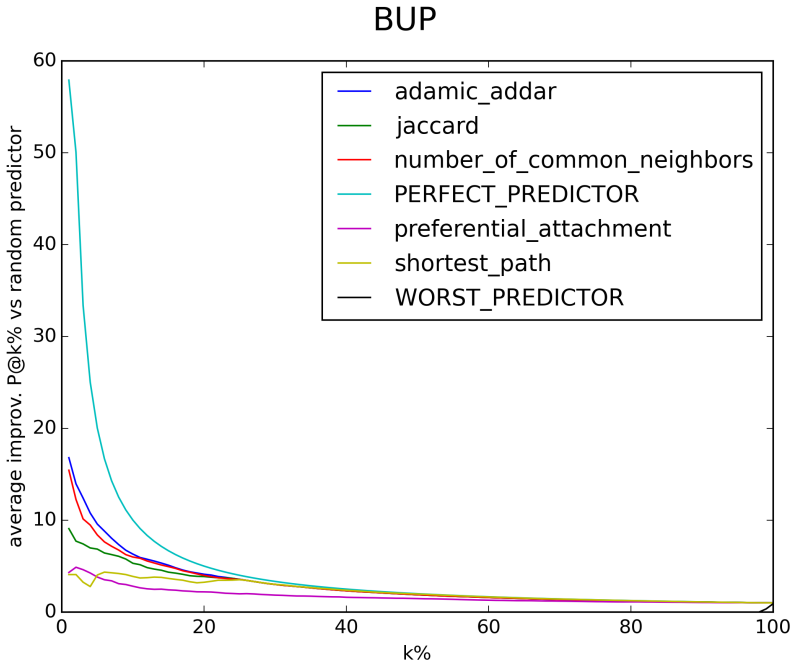
We now show the plots of the average improvement of the P@k compared to a random predictor. For each of the 6 data-set, 7 rankings of predicted links have been made with 7 different methods. For each method the P@k has been computed and then divided by the P@k of a Random Predictor, to measure how better it performs compare to a random choice of predicted links.
The number of i corrected links picked by the random predictor from k trials, has the same probability of an hyper-geometric distribution:

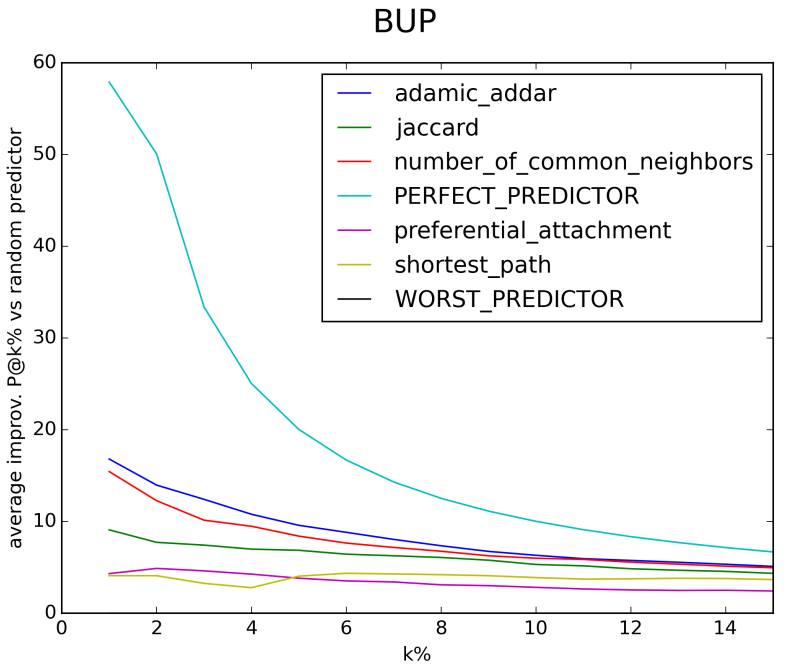
$$i \sim H(n, c, k) \to \mathbb{E}(i) = \frac{c}{n}\, k$$

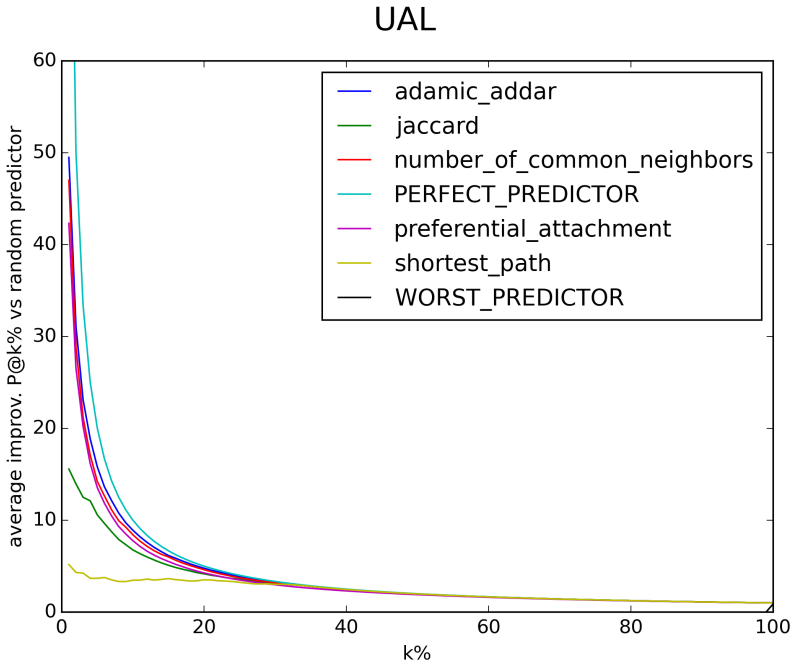
where n is the number of total edges to analyse, c is the number of links of the test graph (the number of links to predict correctly), and k is the number of ones we draw from the top of the ranking of the random predictor, which actually is a random list. We will divide then every P@k of an actual predictor by the *Random P@k* defined as follow:

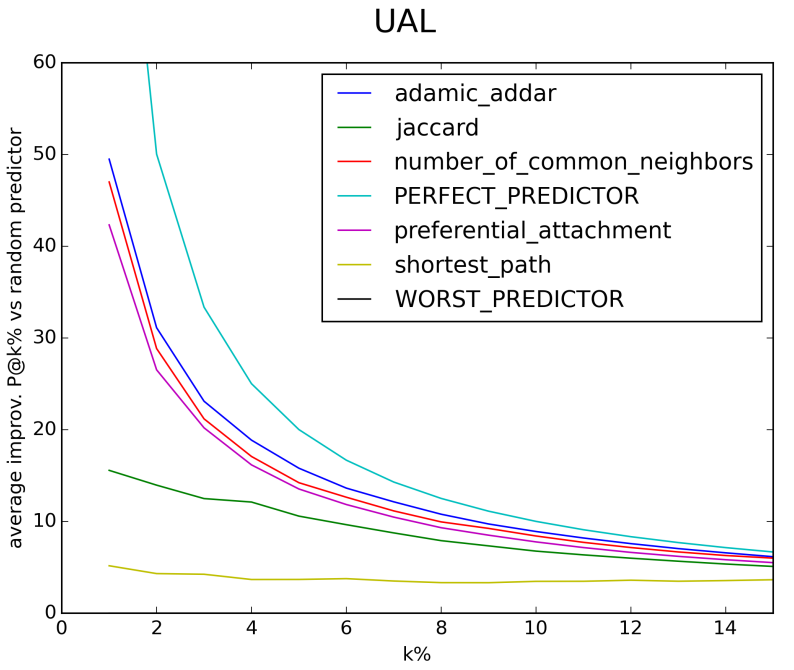
$$Random\ P@k = \mathbb{E}\left(\frac{i}{k}\right) = \frac{c}{n}$$

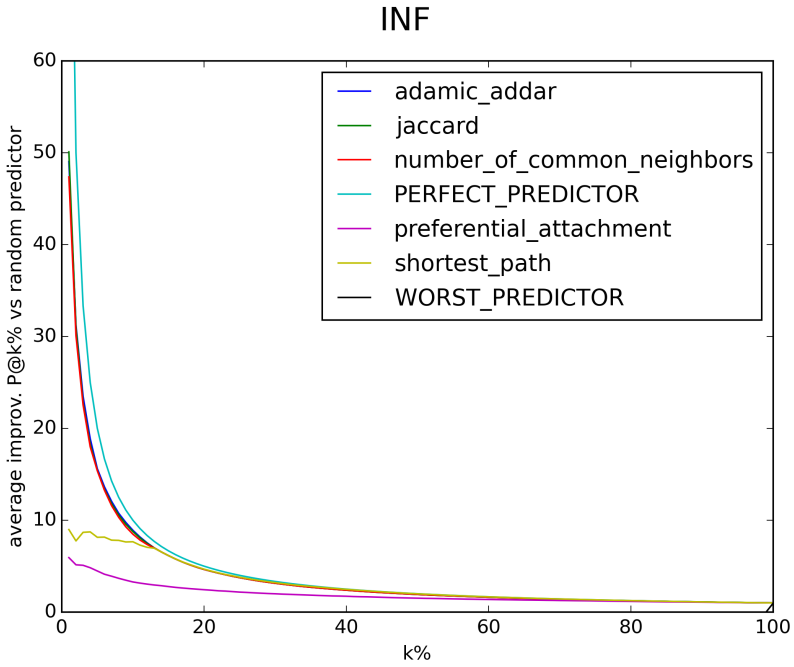
$$P@k\ increment = \frac{P@k_{actual\,predctor}}{c/n}$$

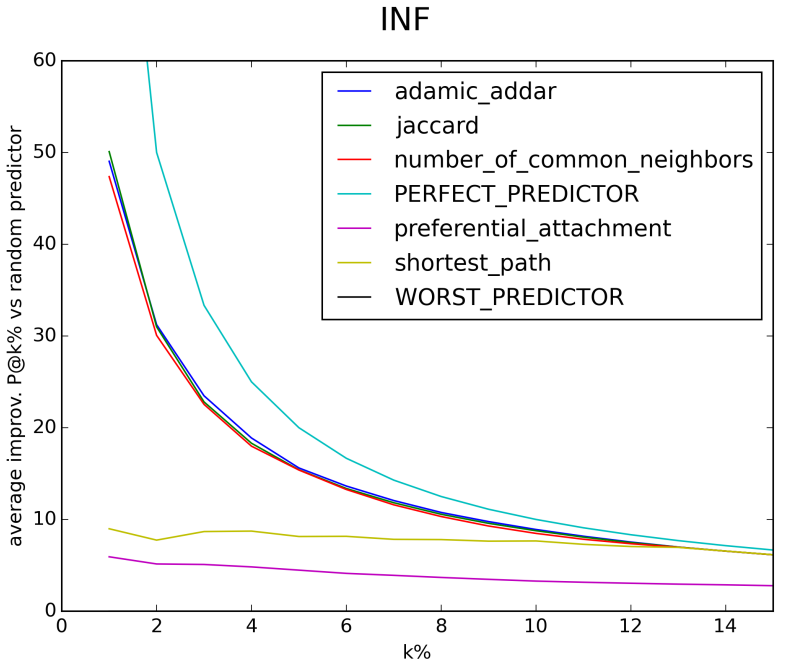

BUP
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100

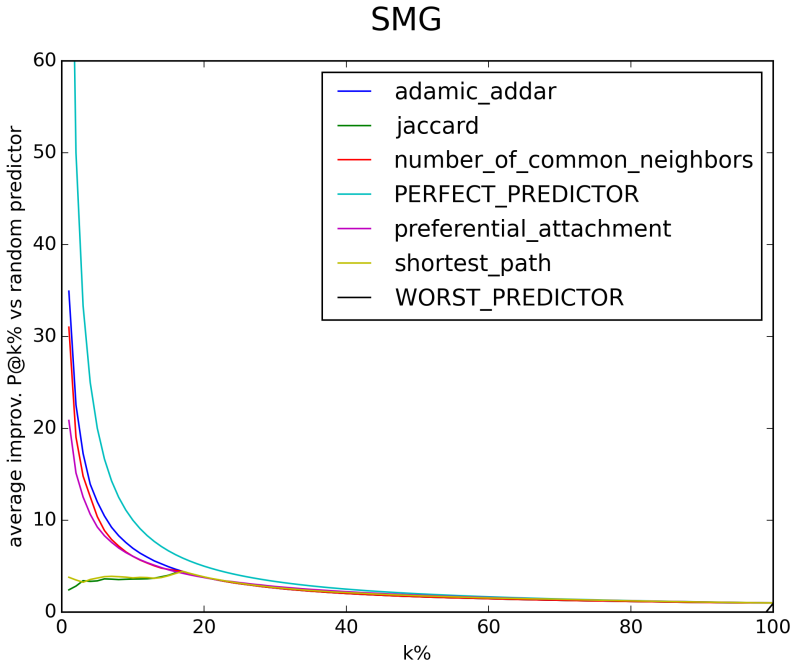

BUP
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14

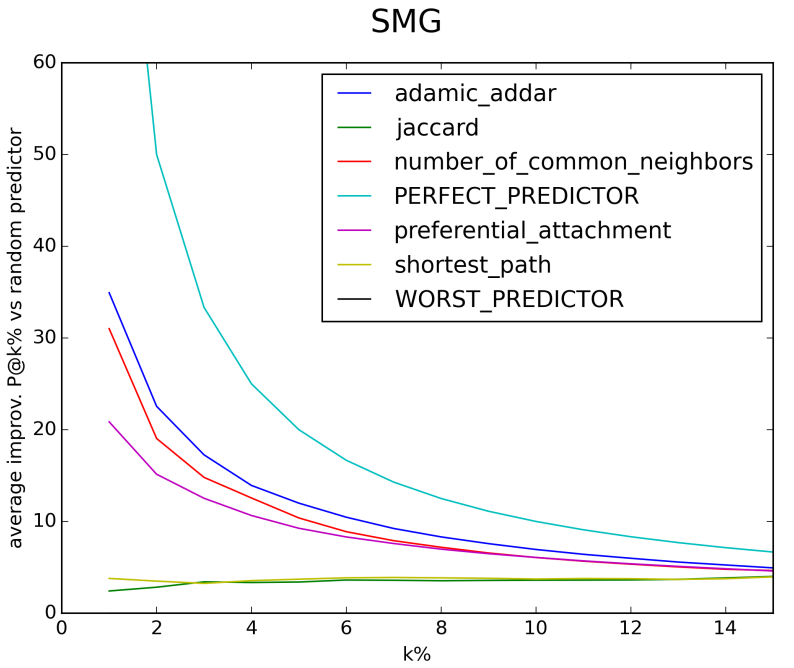

UAL
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100

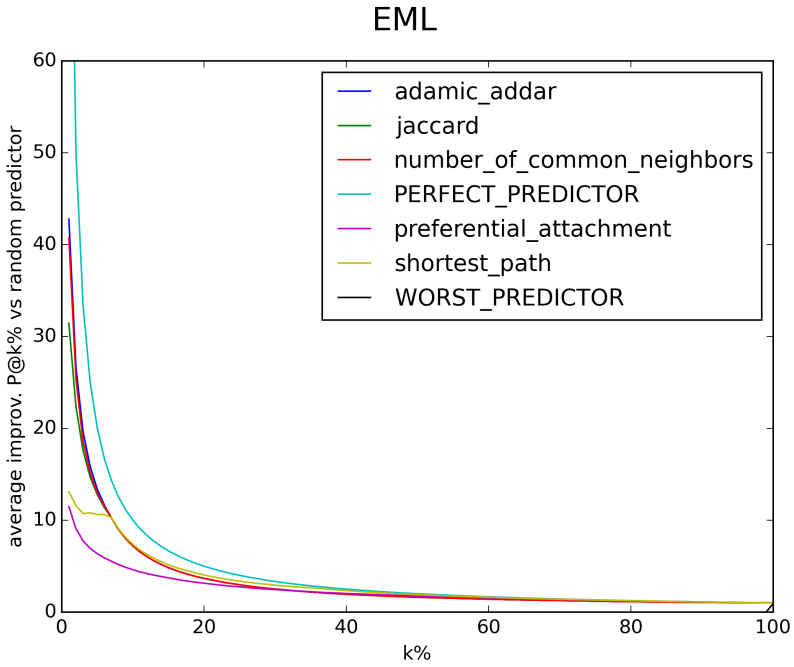

UAL
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14

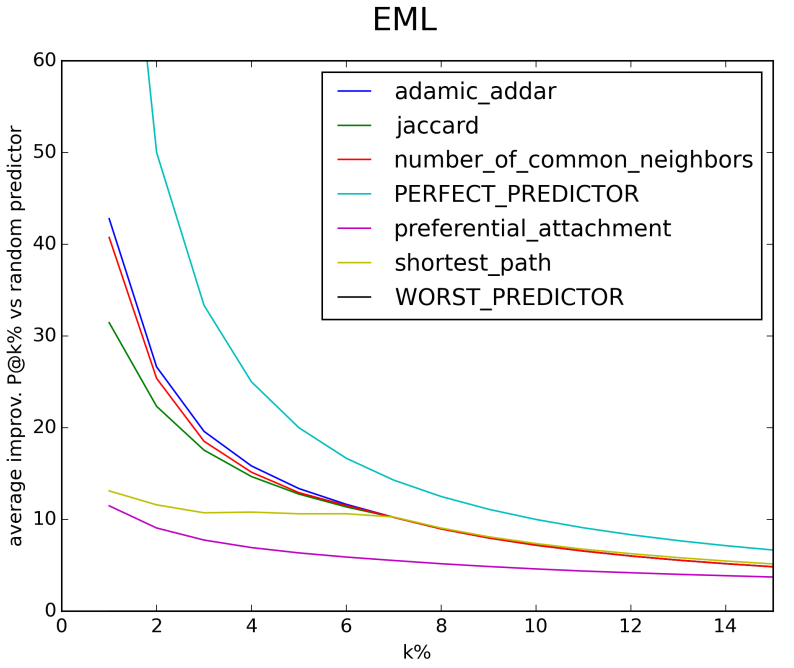

INF
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100

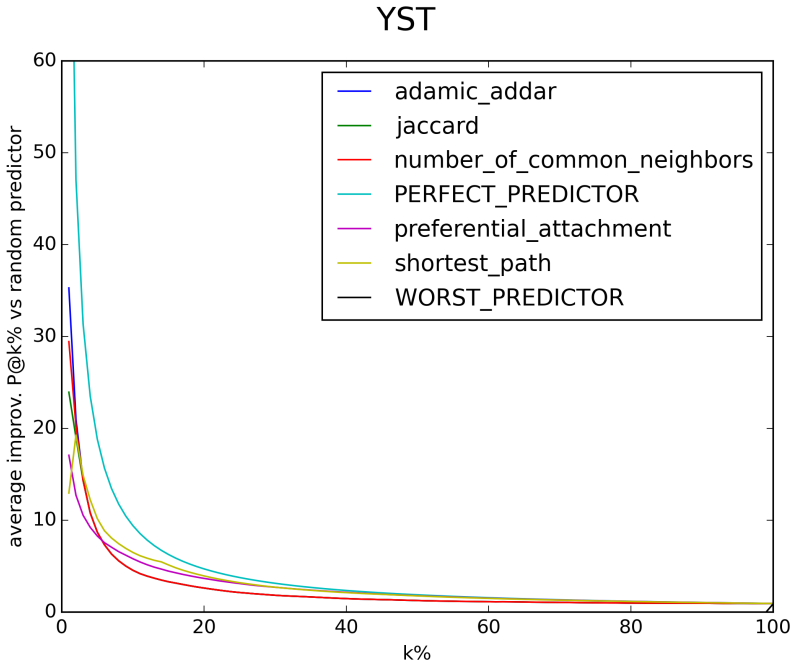

INF
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14

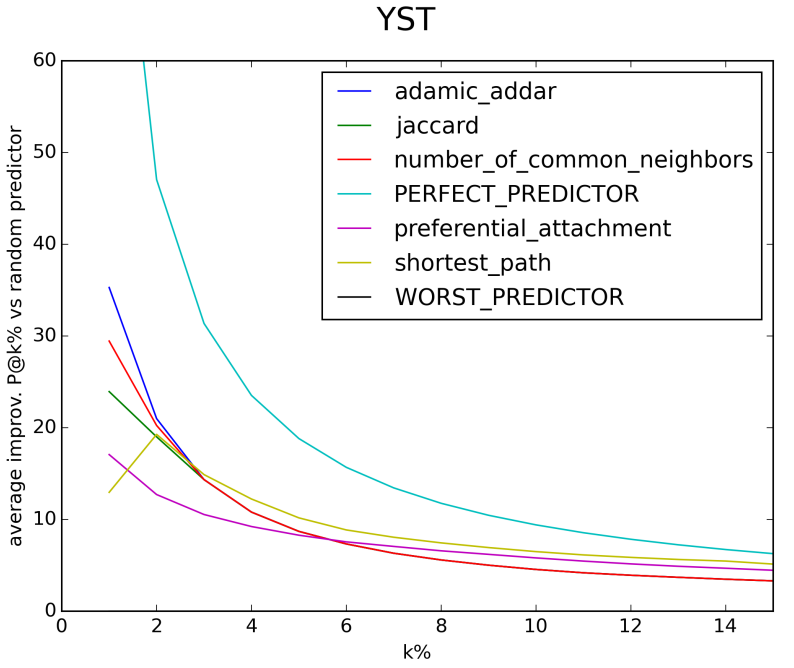

SMG
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100


SMG
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14


EML
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100


EML
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14


YST
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
20
40
60
80
100


YST
adamic_addar
jaccard
number_of_common_neighbors
PERFECT_PREDICTOR
preferential_attachment
shortest_path
WORST_PREDICTOR
average improv. P@k% vs random predictor
k%
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14

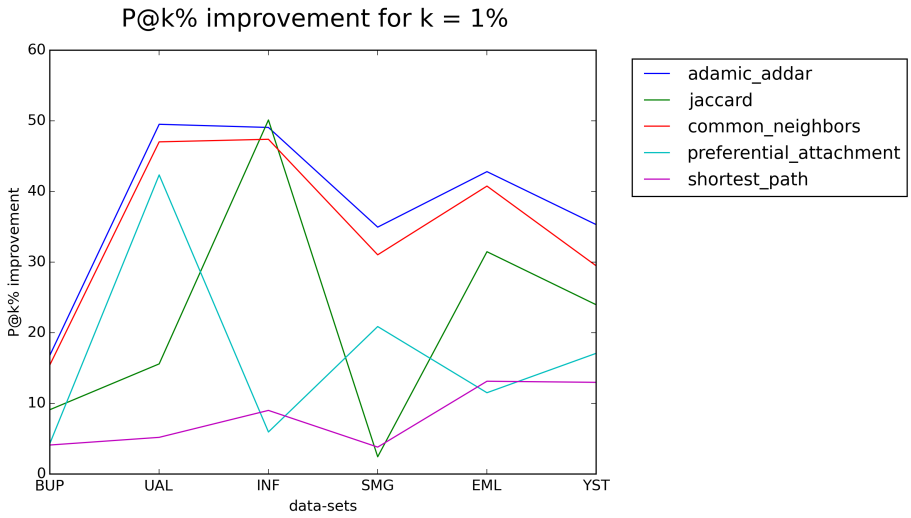
4 Answers

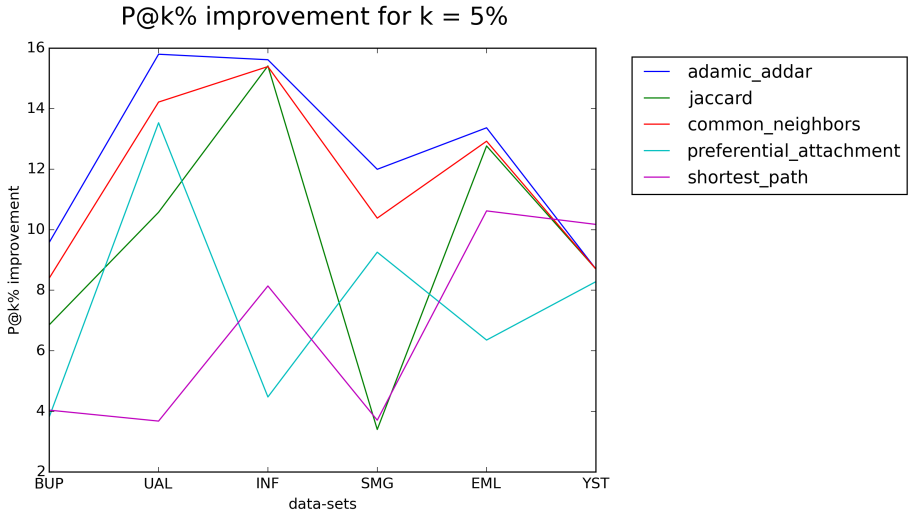
- The formula to compute the total number of pairs of nodes to analyze, a, is:

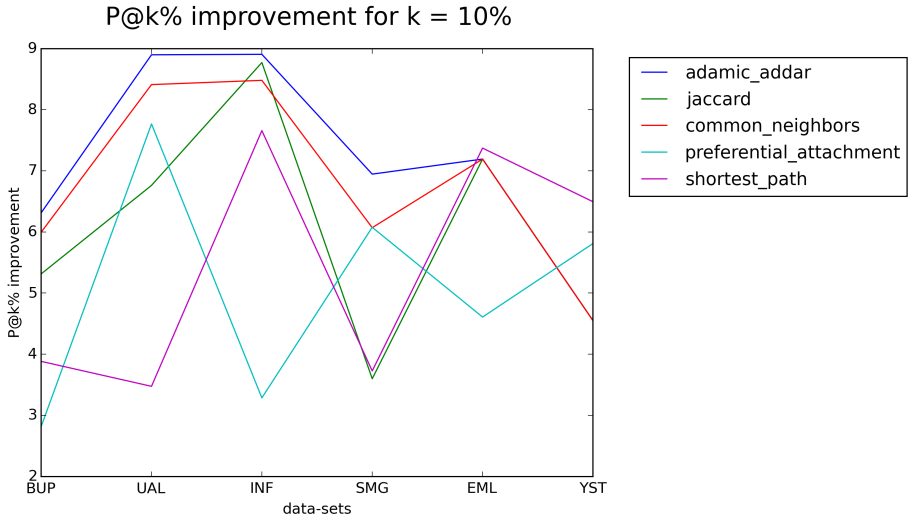
$$a = \binom{n}{2} - e$$

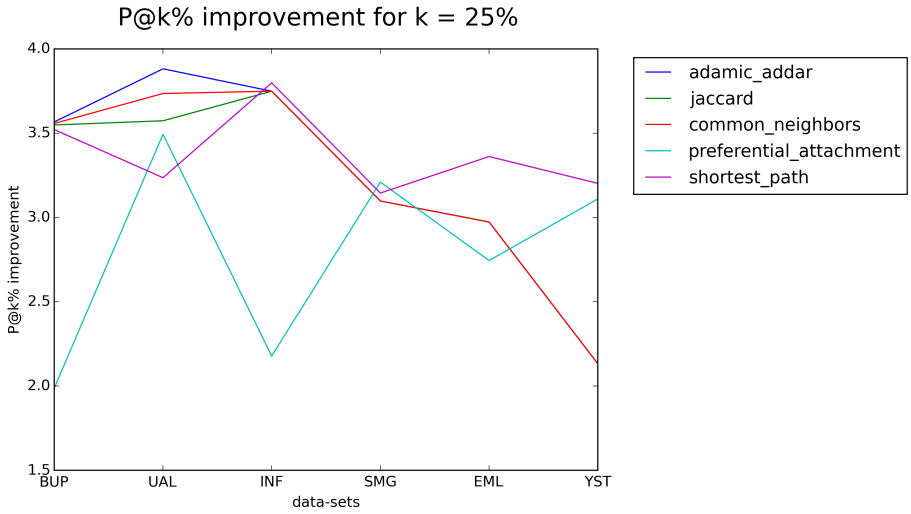
  where n and e are the number of nodes and edges in the training graph.

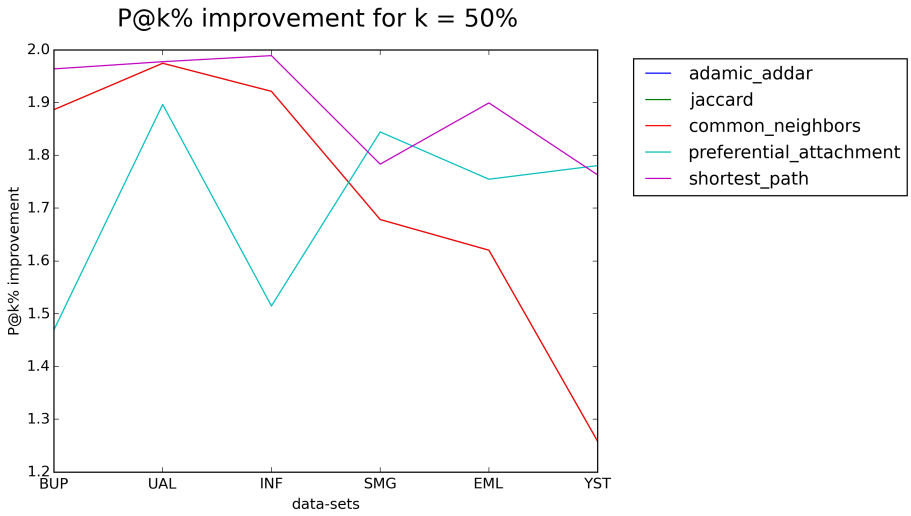
- The performance depends on the topology of the network of each data-set. Each data-set doesn't just have a different size, but it might also have different density, clusters, diameter and so on. Furthermore it might represent scenario where not all methods can applied because of the theories behind them. For example the assumption "rich get richer" of the preferential attachment might be wrong in some social graph and right in another.
  To really assess the real performance of those methods we should take into account all those different features. Anyway few statements can be done thanks to those plots.


P@k% improvement for k = 1%
P@k% improvement
data-sets
BUP
UAL
INF
SMG
EML
YST
0
10
20
30
40
50
60
adamic_addar
jaccard
common_neighbors
preferential_attachment
shortest_path


P@k% improvement for k = 5%
P@k% improvement
data-sets
BUP
UAL
INF
SMG
EML
YST
2
4
6
8
10
12
14
16
adamic_addar
jaccard
common_neighbors
preferential_attachment
shortest_path


P@k% improvement for k = 10%
P@k% improvement
data-sets
BUP
UAL
INF
SMG
EML
YST
2
3
4
5
6
7
8
9
adamic_addar
jaccard
common_neighbors
preferential_attachment
shortest_path


P@k% improvement for k = 25%
P@k% improvement
data-sets
BUP
UAL
INF
SMG
EML
YST
1.5
2.0
2.5
3.0
3.5
4.0
adamic_addar
jaccard
common_neighbors
preferential_attachment
shortest_path



We can state then that:

- For small k the best methods are generally Adamic-Addar and Number of common neighbours.

- For big k the best method becomes generally Shortest Path while Jaccard, Adamic-Addar and Number of common neighbours merge in one another.

- Generally the smaller graphs, BUP, UAL and INF, are better predicted by Adamic-Addar and Number of common neighbours, while the bigger ones EML and YST are working better with Shortest Path and Preferential Attachment for k big enough.

- Jaccard is performing great just with INF and terribly with SMG, while Preferential Attachment is doing the opposite.This means that they must really depend on the inner features of the graph, since they performed completely differently in different graphs. Also Shortest Path presents an indecisive behaviour.

– To conclude, we might be interested in methods performing great on different data-sets for small k so they advice correctly the top rank predicted links, in this case Adamic-Addar is the best method. For the same reason the worst method is Preferential Attachment, which depends a lot on the kind of graph and it performs better just when guessing with a large k of possible links.

– Other observation can be made on those graph, but in the end it is always better to choose a method for a graph by trying them all, rather then relying on past experiences, especially without taking into account what the graphs represents and how the nodes usually interacts with one another.