Contents

1 Data Loading and Processing

Steps taken in preprocessing

The graph that is loaded from **facebook.txt** has many self loops and duplicate edges (a,b) and (b,a). Such edges are to be removed (as discussed with sir). We also will need a validation dataset, which will be needed while training the logistic regression.

In the sections where we have to perform link prediction using heuristics, I have divided the input graph **G** into two parts G_{train} (80% edges) and G_{test} (20% edges), but for the logistic regression part, I have further divided the graph **G** into G_{train} (60% edges), $G_{validation}$ (20%edges) and G_{test} (20%edges). This validation dataset will be used only for finetuning parameters (like the β in katz measure) as well as for training the logistic regresssion (as discussed with sir).

Test Fractions Used

Let us denote test fraction as λ henceforth. The problem statement specifies certain test fractions (say λ) for splitting the data. But, empirically, the algorithm to split the data may give a different test fraction as at every nodes, the x% nodes are sampled, and at the end, these sampled edges are combined in the test set. I have tried to minimise the discrepancy, and to maintain uniformity in the data by allowing only (a,b) node pairs if $a < b$. Here are the test fraction mapping that I have used:

- Desired Test Fraction 0.1 → Used Test Fraction 0.11 → Empirical Value 0.994
- Desired Test Fraction 0.2 → Used Test Fraction 0.21 → Empirical Value 0.1993
- Desired Test Fraction 0.3 → Used Test Fraction 0.31 → Empirical Value 0.3002
- Desired Test Fraction 0.4 → Used Test Fraction 0.41 → Empirical Value 0.4001

Here, the **Desired Test Fraction** is the value specified in the problem statement, the **Used Test Fraction** is the value I used in my code, and the **Empirical Value** is the value that was actually obtained after running the splitting algorithm.

Also, the lectures mention a "K", that is usually given in the problem specification (Top k predictions to be considered). But since there was no such K mentioned in the problem statement, I have set this K to ∞, ie, all score predictions are considered for MRR/MAP (also discussed with sir).

2 Heuristic Based Link Prediction

In this section, we perform 4 heuristic based link prediction methods on the input Graph.

Results: (Task 3)

λ	Evaluation Method	Algorithm			
		Adamic Adar	Common Neighbor	Preferential Attachment	Katz Measure
0.1	MAP	0.776	0.772	0.128	0.752
	MRR	0.878	0.876	0.209	0.858
0.2	MAP	0.726	0.719	0.112	0.698
	MRR	0.875	0.872	0.242	0.860
0.3	MAP	0.689	0.679	0.103	0.659
	MRR	0.870	0.866	0.278	0.852
0.4	MAP	0.644	0.631	0.097	0.616
	MRR	0.856	0.852	0.313	0.838

Evaluations and Analysis

This subsection has all the analysis/evaluation done as required in the problem statement.

- **(Task-4):** According to the results we obtained, **Adamic Adar** seems to give the best results. Although the results obtained using common neighbours are comparable with those obtained using Adamic Adar, the former is indeed, is a bit inferrior as compared to the latter.

  Common Neighbors simply consider the intersection of the adjacent edges of the given query node-pair. This is highly susceptible to getting biased by "celebrity nodes" or "popular nodes". Adamic Adar, on the other hand, tries to supress these "celebrity nodes" by weighting the contribution of each common neighbor by the degree of the common neighbor. So basically, if a "socially inactive node" is a common neighbor between some two nodes (u,v), then the contribution of this common neighbor should be higher as compared to a celebrity node.

  Katz Measure also has a decent MAP score, but it still is inferior to Adamic Adar, as Adamic Adar relies on local features of a graph (ie the common neighbors) but Katz Measure relies on the number of paths between two nodes. Again, Katz Measure is susceptible to paths via "celebrity nodes". Preferential Attachment has a really low performance as it is based on the concept "socially active nodes tend to create more edges". This is true to some extent. A socially active node will try to be friends with some nodes which are in it's vicinity (like Common Neighbours), but it is improbable that an edge can be established between any two random nodes, just because they are "very socially active". For example it is absurd to say that two socially active people, one in Russia and one in India, should be friends on facebook.

- **(Task 5):** From the results collected above, we observe that MRR offers a greater score (by magnitude). But, **MAP is actually much superior** to MRR, since MRR just looks at the first edge predicted correctly by a link prediction algorithm.

Hence it is very easy to fool MRR, even for a poor algorithm. But MAP, on the other hand, is much stricter. It considers the average precision for each node, and appropriately penalizes all the mispredictions. (unlike MRR which harshly penalizes all mispredictions untill the first correct prediction, which has a lot of uncertainty to it). So basically, an algorithm that performs well on MAP will mostly perform well on MRR, but the converse is not necessarily true. Hence I conclude that MAP is better than MRR.

- **(Task6):** Yes, the results that we get using MAP or MRR, for various link prediction algorithms are consistent. This is actually expected and can be explained. As we stated in the point above, if a method performs well using MAP, then we can expect it to perform well using MRR as well. Hence all the algorithms that are giving a high MAP score are performing well with respect to MRR as well.

3 Logistic Regression Based Link Prediction

This section summarises the observation and results obtained by training a logistic regression using Lasso Stochastic Gradient Descent, on the features obtained in the previous section.

Results: (Task 7)

Evaluation Method	Test Fraction			
	0.1	0.2	0.3	0.4
MAP	0.759	0.694	0.647	0.596
MRR	0.863	0.849	0.837	0.817

Evaluations and Analysis

This subsection has all the analysis/evaluations as required in the problem statement.

- **(Task 8):** I believe that the logistic regression predictor fares better than the heuristic based link prediction algorithms that we implented in the previous section. The logistic regression considers a weighted sum of the scores from all the algorithms. So in a way, it is extracting the best of all algorithms byt fine tuning the weights and giving us a combined result which has a better performance than the individual predictors. Although the MAP/MRR figures of logistic regression are slightly lower than the heuristic based methods, we also must consider that in the 60-20-20 division of our graph, only 20 percent edges were used for training the weights (which is much much less than what the heuristic based methods considered for training). Hence, overall, logistic regression seems to be a better method.

4 Usage of Code

1. The following files are included in my submission directory.

   - **run.sh:** A bash script to fit the problem statement (to have the 1st argument as data path). But it will be more convenient to run the python script directly.
   - **run.py:** The main code file which has the driver code, but needs the data path as a flag argument.
   - **classes.py:** Has the defination of classes for a Graph object and a Logistic Regression object.
   - **utils.py:** Has utility functions that are called in both classes.py and run.py. The functions that are defined here are - MRR, MAP, common_neighbors, katz_measure, find_nbr_nonnbr
   - **test_fraction_tuner.py:** Contains the code that I used to fine tune the test fraction for various desired values

2. Python libraries necessary:

   - networkx
   - pickle
   - matplotlib
   - sklearn
   - numpy
   - argparse,

3. Default Running the code. Please note that the entire execution of this code (including logistic regression) takes around 20 minutes. I thank you for your patience.

```
$ ./run.sh <path to dataset>
# This runs the code with test fraction = 0.2 for all the 4 link
   predictors and Logistic Regression.
```

4. Running the python code. Please note that sys.argv and argparse cannot be used together. Hence, since I already had used argparse for various arguments, I could not have the second argument fixed as dataset path. Hence I have added a –data flag which can take the path to the dataset. Thanks again for considering this. (else the bash script satisfies the problem statement, although the python script is more convenient)

```
$ python3 run.py --data <path to dataset>
# This runs the default code on the dataset specified by the flag.
```

5. Specifying a test fraction

```
$ python3 run.py --test_fraction <!!>
# This runs the code with the custom test fraction. The only valid
   input for this argument are {0.1, 0.2, 0.3, 0.4}. Any other
   input will raise a KeyError.
```

6. Running a specific algorithm

```
$ python3 run.py --only_adamic_adar
$ python3 run.py --only_preferential_attachment
$ python3 run.py --only_katz_measure
$ python3 run.py --only_common_neighbors
$ python3 run.py --only_logistic_regression
# This runs the code with the specified algorithm. This flag can be
    combined with the flag for specifying a test fraction
```

7. Saving and Loading the score predictions in a pickle format

```
$ python3 run.py --dump_pickle
$ python3 run.py --load_pickle
# This runs creates a directory './pickles' and saves the
   predictions in this folder. These pickles can subsequently be
   loaded to give the MAP/MRR analysis faster.
```