I. Feature Extractions:

A. Feature base:

a. Edge features:

i. Common neighbors: $|\Gamma(\mathrm{u}) \cap \Gamma(v)|$

ii. Jaccards coefficient: $\frac{|\Gamma(\mathrm{u}) \cap \Gamma(v)|}{|\Gamma(\mathrm{u}) \cup \Gamma(v)|}$

iii. Adamic Adar: $\sum_{\mathrm{z} \in \Gamma(\mathrm{u}) \cap \Gamma(v)} \frac{1}{\log(|\Gamma(z)|)}$

iv. Preferential attachment: $|\Gamma(\mathrm{u})| \times |\Gamma(y)|$

v. Commute time: $\frac{1}{\Gamma(\mathrm{u})} + \frac{1}{\Gamma(v)}$

vi. Shortest path: $\frac{1}{length(shortest_{path(u,v)})}$

vii. Katz score: $\sum_{\mathrm{L}=1}^{\infty} \beta^L A^L = (\mathrm{I} - \beta \mathrm{A})^{-1} - I$

b. Node features:

i. degree: $\Gamma(\mathrm{u})$

ii. 1/degree: $\frac{1}{\Gamma(\mathrm{u})}$

iii. numeric features (4 features): 年齡、完成度百分比、性別、公開

iv. categorical feature (location): 地區

B. Feature creation:

a. Edge features:

i. Common neighbors, Jaccards coefficient, Adamic Adar, preferential attachment, and commute time can be calculated by linearly comparison through neighborhood lists for node u and node v. The amortized time complexity is $\mathrm{O}(\mathrm{instance} \times \mathrm{E})$.

ii. Time complexity to calculate shortest path from node u to node v is $\mathrm{O}(\mathrm{N} + \mathrm{E})$, so the naïve complexity for the whole problem is $\mathrm{O}\big(\mathrm{instance} \times (\mathrm{N} + \mathrm{E})\big)$. However, if the order of feature creation is well planned, by using the single source BFS shortest path algorithm, the overall time complexity is $\mathrm{O}\big(\mathrm{N} \times (\mathrm{N} + \mathrm{E})\big)$.

iii. Katz score needs to do a large matrix inversion, which is impossible in terms of time complexity. Thus, we imply an approximation of the algorithm by ...

b. Node features:

i. Numerical data have few missing values. After performing simple imputation algorithm, we can use these four features directly.

ii. Categorical data is poor for this task. Almost all categorical features have more than 50% missing rate, which make this problem much challengeable. At the end, we just use the "地區" feature after performing binary expansion on it, which results in about 200 features.

II. Dataset creation:

A. TRIAN and TEST:
We sample about three times of not-existent edges as negative instance and uses these edges to create TRAIN and TEST features. All the positive and negative edges are the instances for TRAIN. For TEST, we enumerate all non-existent possible edges as instances. That is to say, there are about $\mathrm{O}(\mathrm{N} \times \mathrm{E})$ instance for TEST.

B. SUBTRAIN and VALID:
We sample 80% edges randomly from TRAIN to create SUBTRAIN and VALID features. These sampled edges are the instances for SUBTRAIN and the left are the instances for VALID.

III. Model:

A. To deal with such large matrix, said 10^9 instances with about 400 features, we only use `Logistic Regression Classifier with L2 norm Regularization` in `LIBLINEAR` as model. However, it still takes a lot of time to train the model and predict labels.

IV. Difficulties:

A. We try hard to create a good validation set in order to give us a direct insight into the TEST performance. To be clear, we hide 20% of edges when creating features for SUBTRAIN and VALID, then judge whether these edges exist or not as the ground truth for validation set. This setting seems very reasonable; nevertheless, we get a huge performance gap between VALID and TEST.

There may be a huge inconsistence between validation set and test set, but we have no way to know.

	TT	TF	FT	FF	PRED	RECALL	F1-SC	MAP
SUBTRAIN	0.20552	0.04446	0.03826	0.71177	0.84306	0.82214	0.83247	0.56424
VALID	0.20297	0.04712	0.03799	0.71192	0.84234	0.81157	0.82667	0.56430
TRAIN	0.17754	0.07245	0.02066	0.72934	0.89577	0.71017	0.79225	0.55525
TEST	– – –	– – –	– – –	– – –	– – –	– – –	0.07036	0.09360

B. Because of the task property, if we want to transform this task into classification problem, we have to sample all possible edges, in other words, non-existent edges, linked with given test nodes. That is to say, the test matrix will be as large as $10^9 \times 400$, and it is extremely time-consuming to create features or predict on this matrix.

   At the end, the feature files take accounts for about 800 GB in terms of disk storage. To solve this problem, we have to put a lot of efforts on the management of disk space and memory space, for example, separated feature files, separated prediction files, and lots of fork usage.