Challenge #2: l'illusion de données personnelles cachées

ELU 501 Data science, graph theory and social network studies

Xuanqi HUANG & Malick TUO

Introduction

Nous sommes des data scientist travaillant chez LinkedIn. Dans le souci de promouvoir leur restaurant dans le baie, nos collègues du pôle marketing ont sollicité notre aide afin de trouver des personnes influentes dans le réseau social LinkedIn, qui représentent l'image du restaurant.

Pour mener à bien cette mission nous disposons d'un jeu de données du réseau LinkedIn au travers duquel nous devons extraites des informations nécessaires à la promotion du restaurant, c'est-à-dire trouver les 5 influenceurs de ce réseau. Ces données sont sous la forme d'un graphe ou les nœuds représentent les personnes dans le réseau, et les liens représentent les relations qu'ils ont entre eux. Ces relations sont traduites par trois attributs qui sont *la* ***localisation*** des personnes ou leurs emplacements géographiques, leurs ***emplois*** c'est-à-dire le/les entreprises ou elles ont travaillé et enfin *le* ***collège*** ou chaque personne du graphe a étudié.

Cependant, cette tâche ne sera pas aussi simple. En effet certaines personnes sur LinkedIn ont omis d'enregistrer certaines informations sur leur profile. On constate qu'il y'a par exemple des personnes qui n'ont pas renseigné leurs localisations ou bien leurs entreprises ou qui n'ont enregistré aucune information sur leur profile. Au niveau du graphe nous nous retrouvons avec des nœuds avec des attributs manquants ce qui complique l'identifications des 5 influenceurs puisque nous manquons d'informations sur les relations entres personnes.

Par ailleurs nous disposons d'un algorithme (l'algorithme 'naïf') qui se base sur des calculs de probabilités et d'hypothèses sociologiques pour prédire les informations manquantes des utilisateurs. Cependant le taux de prédiction étant faible (environ 30%) nous ne pouvons pas nous baser sur ces informations pour trouver les influenceurs du réseau.

En sommes notre travail peut-être subdiviser en trois grands lots :

- Etude des données manquantes et du methods existant pour mieux comprendre ce que nous devons faire pour améliorer la résultat.
- Proposer un algorithme pour prédire les attributs manquants des utilisateurs en faisant des hypothèses fondées sur la sociologie et des calculs de probabilité tout en sachant que le taux de prédiction de notre algorithme doit être supérieurs à celui de l'algorithme naïf ;
- Trouver sur la bases des valeurs prédites les 5 personnes du réseau LinkedIn les plus influentes, et dont les influences se trouve dans la baie. Nous étudierons plusieurs solutions puis ferons des comparaisons afin de définir celle que nous

jugeons être la plus efficace. Notons que la véracité de nos informations produites dans le lot 2 dépendra de la qualité de nos prédictions.

Pour mener à bien notre étude, nous utiliserons la méthodologie CRISP-DM

1. Compréhension des données et méthode existante

1.1 Etude des données manquantes

Chaque nœud du graphe représente une personne sur LinkedIn. Chaque personne doit avoir exactement 3 attributs (location, employeur, collège). Les attributs peuvent avoir une ou plusieurs valeurs ; par exemple pour un utilisateur donné si son attribut employeur existe il contient la liste des entreprise fréquentées. L'entreprise la plus récente dans la liste représente l'entreprise actuelle où se trouve l'utilisateur.

A partir du graphe nous avons extrait les informations présentent dans le tableau suivant :

Intitulés	**Taux**	**Pourcentage d'informations recueillies %**
Nombre de personnes (nœuds) dans le graphe	811	100
Nombre de personne ayant spécifié leur localisation	336	41,43
Nombre de personne ayant spécifié leur employeur	297	36,62
Nombre de personne ayant spécifié leur université	230	28,36
Nombre de personne ayant renseigné les trois attributs	225	27,74
Nombre de personne ayant renseigné seulement deux attributs	77	9,49
Nombre de personne ayant renseigné qu'un seul attribut	35	4,31
Nombre de personne ayant renseigné aucun attribut	475	58,56

On constate que plus de la moitié des personnes sur LinkedIn (475) n'ont pas renseigné d'informations sur leurs profile et seulement un faible nombre de personne (225) ont enregistré toutes les informations les concernant. Il serait alors difficile de trouver des influenceurs avec seulement les personnes dont les informations sont complètes. D'où la nécessité de trouver un algorithme pour prédire les attributs manquants.

1.2 Etude de la méthode existante (naiv_methods)

Cette méthode fait la prédiction en supposant qu'un noeud partage potentiellement les mêmes attributs avec ses voisins. Mais il y a 60% des données manquantes, Donc il peut notamment avoir le cas suivant: il peut exister des noeuds, dont les attributs sont manquants, qui ont des voisins qui ne possèdent pas d'attributs non plus. Dans ce cas là, ce n'est pas possible de prédire ses attributes.

De plus, les prédictions se font avec seulement les voisins des noeuds. Ces prédictions ne peuvent pas bien représenter l'attribut des noeuds lorsqu'ils ont très peu de voisin. Cela va généralement conduire à un problème de précision.

Enfin, la répartition des voisins n'est pas précise du tout. Parce que ceux qui ont deux ou trois voisins pourraient avoir des valeurs d'attributs totalement différents de ceux de ses voisins. Et ça arrive souvent dans ce graphe.

Afin de mieux prédire les attributs manquants, Il faut d'abord trouver des relations cachées entre les noeuds. Cela nous permettra d'avoir plus d'indices à trouver l'attribut d'un noeuds sur tout pour les noeuds qui n'ont pas assez de voisins. Ensuit, nous devons trouver une autre méthode de répartition du graphe afin de mieux trouver les communautés dans ce réseau social. En fin, nous devons aussi assurer que tous les attributs manquants seront prédit.

2. Notre stratégie pour la prédiction des profils

Dans cette partie, nous allons vous présenter nos solutions pour prédire les attributs manquants:

2.1 Random Forest Regression:

Au tout début de ce challenge, nous avions opté pour l'utilisation de l'algorithme "Random Forest" pour la prédiction. Après avoir étudié les données, nous avons constaté qu'il y a 89 locations différentes , 100 collèges différents et 227 emplois différents. Il y a trop de classe. Sachant que le but du projet est de trouver les cinq influenceurs à la Baie, nous faison une prédiction de location concernant seulement à la Bay Area. Supposant que ceux qui habitent à la Bay Area ont pour attribut de location '1', si non l'attribut de location equal '0'.
Voici les résultats que nous avons obtenus :

```python
from sklearn.ensemble import RandomForestRegressor

def set_missing_location(train_x,train_y,cross_x):

    rfr = RandomForestRegressor(random_state=0, n_estimators=2000, n_jobs=-1)
    rfr.fit(train_x, train_y)
    predicted = rfr.predict(cross_x)
    pred_y = np.round(predicted)
    return pred_y, rfr
```

```python
x = df_train[['ngb_location','ngb_employer','ngb_college']].as_matrix()
y = df_train['location'].as_matrix()
cross_x = df_cv[['ngb_location','ngb_employer','ngb_college']].as_matrix()
cross_y = df_cv['location'].as_matrix()
predicted,rfr = set_missing_location(x,y,cross_x)
```

```python
a = evaluation(predicted,cross_y)
print("precision of the prediction for cross_validation_set is %f%%" % a)
```

```
precision of the prediction for cross_validation_set is 94.000000%
```

La précision est élevée. C'est parce qu'il y a très peu des gens qui habitent dans la " Bay Area" (moins '1' dans le cross_y). Le résultat de la prédiction est toujours égal à '0'. De plus, il n'y a pas beaucoup de données qui contiennent tous les attributs, cet algorithme n'est donc pas adapté. Nous devons envisager d'autres algorithmes.

2.2 Prédiction basée sur le graphe

Après avoir lu l'article 'User Profiling in an Ego Network[1]', nous trouvons que l'ego network proposé par cet article n'est pas compatible à notre projet car pour implémenter cette méthode il faut connaître les relations entre les noeuds(collègue, camarade de classe,etc).Dans notre projet, les données manquantes n'ont pas du tout d'attribut. Nous ne pouvons que trouver quelconque relations à partir du graphe. Mais Il y a aussi des idées qui nous inspirent.

2.2.1 Partition de Louvain

Afin d'améliorer la précision de la prédiction, nous devons trouver suffisamment de noeuds pertinents pour chaque noeud à prédire. Après avoir étudié les fonctions intégrées dans networkx, nous avons trouvé le meilleur moyen de diviser un graphe, Louvain Partition.
(le graphe partitionné en annexe n°3, les différents communautés sont en couleurs différents.)

Nous voyons bien que le graphe est divisé en 19 communautés. La plus petite communauté contient 5 personnes.

```
community 1 has 119 people
community 2 has 82 people
community 3 has 79 people
community 4 has 69 people
community 5 has 64 people
community 6 has 60 people
community 7 has 54 people
community 8 has 48 people
community 9 has 45 people
community 10 has 33 people
community 10 has 33 people
community 11 has 31 people
community 12 has 28 people
community 13 has 26 people
community 14 has 15 people
community 15 has 15 people
community 16 has 15 people
community 17 has 10 people
community 18 has 8 people
community 19 has 5 people
```

Pour faire la prédiction, nous supposons que les noeuds qui sont dans une même communauté potentiellement partagent leurs attributs.

```
print the groudtruth rate for location_prediction
51.157895% of the predictions are true
print the groudtruth rate for college_prediction
32.421053% of the predictions are true
print the groudtruth rate for EmployerPrediction
33.473684% of the predictions are true
```

La résultat est notamment plus fiable que celle de la méthode naiv. Nous avons continué d'améliorer cette méthode . Une amélioration s'illustre à la suite.

2.2.2 Amélioration : classifier les noeuds par leur degrés

Comme nous étudions les données manquantes, nous trouvons que des noeuds ont des degrés beaucoup plus élevée que ceux des autres noeuds , leurs prédictions seraient plus fiables. Donc, nous choisissons de diviser les noeuds par leur degrés.

```python
# (i,j) for the degree i ,there are j people having this degree
degrees = dict(G1.degree())
node_high_dgr = []
node_mid_dgr = []
node_low_dgr = []
for i in G1:
    if degrees[i] > 10:
        node_high_dgr.append(i)
    elif degrees[i] >= 4 :
        node_mid_dgr.append(i)
    else:
        node_low_dgr.append(i)
cpt = Counter(degrees.values())
cpt = sorted(cpt.items(),key = lambda t: t[0],reverse = False)
```

Comme indiqué ci-dessus,
Ceux qui ont un degré supérieur que 10 sont dans la classe_high_degrees
Ceux qui ont un degré supérieur que 4 mais moins de 10 sont dans la classe_mid_degrees
Ceux qui restent sont dans la classe_low_degrees
Et Voici les résultats que nous avons eu:

```
prediction length is  30
print the groudtruth rate for location_prediction
83.333333% of the predictions are true
print the groudtruth rate for college_prediction
56.666667% of the predictions are true
print the groudtruth rate for EmployerPrediction
66.666667% of the predictions are true
```

high_degree_class

```
prediction length is  120
print the groudtruth rate for location_prediction
55.833333% of the predictions are true
print the groudtruth rate for college_prediction
38.333333% of the predictions are true
print the groudtruth rate for EmployerPrediction
36.666667% of the predictions are true
```

mid_degree_class

```
prediction length is  325
print the groudtruth rate for location_prediction
46.461538% of the predictions are true
print the groudtruth rate for college_prediction
28.000000% of the predictions are true
print the groudtruth rate for EmployerPrediction
29.230769% of the predictions are true
```

low_degree_class

```
Results for the naive method
print the groudtruth rate for location_prediction
30.736842% of the predictions are true
print the groudtruth rate for college_prediction
25.684211% of the predictions are true
print the groudtruth rate for EmployerPrediction
19.789474% of the predictions are true
```

Naiv_method

Par rapport à la méthode originale, notre méthode a obtenu une meilleure précision.Cela peut potentiellement faciliter notre recherche des cinq influenceurs dans la partie suivante.

3. Recherche des cinq personnes les plus influençants

3.1 Répartition des utilisateurs

L'objectif de cette troisième partie est de trouver 5 influenceurs dans le réseau LinkedIn dont les personnes influencées se situent dans la "bay area". On peut commencer par structurer notre graphe. Comme indiqué dans la partie 2 (prédiction), nous avons fait une estimation des attributs manquants afin de disposer de grandes données pour trouver nos 5 influenceurs. Tout le travail qui va suivre se fera sur la base de ces nouvelles données.

Maintenant vue que les influencées doivent obligatoirement être localisées dans la "bay area".

Voici la construction de graphe_Bay:

```
for i in G:
    influence_matrix[i] = 0
for k in G:
    for ngb in G[k]:
        if ngb in loc_key_matched and ngb not in loc_key_matched_pred:
            edges.append((k,ngb,{'weight':1}))
            influence_matrix[k] += 1
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred1:
            edges.append((k,ngb,{'weight':proba1}))
            influence_matrix[k] += proba1
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred2:
            edges.append((k,ngb,{'weight':proba2}))
            influence_matrix[k] += proba2
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred3:
            edges.append((k,ngb,{'weight':proba3}))
            influence_matrix[k] += proba3
G_Bay.add_edges_from(edges)
```

On commence par recenser seulement les personnes (les influenceurs) dans le graphe dont les *voisins directs* (les influencées) se trouvent dans cette zone. On obtient une nouvelle liste de personnes à partir de laquelle nous allons élaborer notre stratégie.

3.2 Graphe et mise en place d'une stratégie

Nous avons implémenté deux algorithmes afin de trouver les 5 influenceurs. ils s'agit du calcul du degré sortant pour chaque noeud et du calcul du poids des liens entre les noeuds. L'idée étant de recenser les noeuds ayant les degrés sortants les plus élevés ce qui implique qu'il s'agit des noeuds ayant le plus de relations (donc une forte influence).
Pour cela, nous avons travaillé avec un graphe orienté. l'orientation des liens entre deux noeuds (A et B) données est définie comme suite:

- si le noeud A est en dehors de la "bay area" a pour voisin le noeud B qui se trouve dans la zone, alors l'orientation de lien est de A vers B (A ---> B).
- si les deux noeuds voisins se trouvent dans la "bay area", alors l'orientation se fait dans les deux sens: A ---> B et B ---> A.

Ainsi nous pourrons calculer le degré sortant de chacun des noeuds et trouver les 5 noeuds qui ont le plus d'influence.

Cependant il faut considérer le fait que les relations entres les noeuds n'ont pas la même force. En effet il y' a des relations qui existaient avant (qui n'ont pas été prédites) et d'autres qui ont été prédites. Vue que taux de prédiction n'est pas égale à 100% cela implique que ces liens présentent une grande probabilité d'erreur et qu'ils ne peuvent pas avoir le même poids que les liens vrais (qui n'ont pas été prédits). Il faut alors affecter des poids aux différents liens pour assurer une certaine corrélation.
<u>affectation des poids au liens</u>

soit A et B deux noeuds. L'affectation des poids au liens se fera comme suite:

- Si A et B sont voisins et que le liens entre ces deux noeuds n'a pas été prédit, le poid du lien est de **1;**
- Si A et B sont voisins et que le liens entre ces deux noeuds a été prédit alors le poids du lien est celui de la **probabilité de prédiction**
- Vue que nous avons distingué au niveau de la prédiction des noeud avec des degré fort, moyen, et faible l'affectation du poids au lien va donc dépendre du degré du noeud.

finalement pour chaque noeud, on calcul le combiné des poids de tous ses liens sortants.

3.3 Résultats

Voir le graphe en **annexe n°1 et annexe n°2;**

Le graphe présente 4 catégories de noeuds à savoir:

- les **noeuds en rouge** représentent les 5 personnes ayant les degrés sortants les <u>*plus élevés*</u> mais dont le combiné des poids les liens est <u>*peu élevé*</u>;

- les **noeuds en bleue** représentent les 5 personnes ayant les degrés sortants les *plus élevés* et dont le combiné des poids les liens est lui *aussi élevé*;
- les **noeuds en noire** représentent 5 personnes ne figurant pas dans le top 5 des noeuds ayant les degrés sortants les plus élevés mais dont le combiné des poids les liens est élevé;
- les **noeuds en jaune** représentent les autres.

On obtient :

- les 5 personnes ayant les degrés sortants les plus élevé: 'U27287', 'U7024', 'U4562', 'U8670', 'U15267'

- le top 5 des personnes ayant le cumule des poids de leurs liens élevé (cumule,personne): (100.671, 'U27287'), (49.988, 'U7024'), (25.852, 'U8670'), (24.176, 'U11566'), (23.346, 'U22747')

Si on considère être plus important les noeuds ayant les plus haut degrés sortants, et que ceux ayant le cumule des poids des liens d'une importance moyenne alors nous choisirons comme influenceurs les noeuds en jaune. cela sous entend que les 5 influenceurs sont les 5 personnes ayant le plus de relations dans la "bay area" mais qui ne sont pas très bien connues de tous.

Sinon si nous considérons que le degré sortant et le poids des liens ont pratiquement la même importance alors nos 5 influenceurs seront les noeuds en bleue. cela sous entend que les 5 influenceurs sont les personnes ayant le plus de relations et qui sont très bien connues de tous.

4. conclusion

4.1 avantages:

Pour la prédiction, notre stratégie a résolu les trois inconvénients de la methods naiv dont l'idée principale est simple. C'est de chercher d'autant de nœuds de référence que possible.

Pour la recherche des cinq influenceurs, notre solution est plus intuitive et facile à réutiliser.

4.2 inconvénients:

Nous avons un défaut principal. Si les attributs des noeuds ont des valeurs similaires ou bien leur entreprises sont les mêmes mais avec une écriture différente, notre algorithme ne peut pas les distinguer.

]:
```
a=[]
for i in groundtruth_location:
    if groundtruth_location[i][0] not in a:
        a.append(groundtruth_location[i][0])
b=[]
for i in location:
    if location[i][0] not in b:
        b.append(location[i][0])
print('%d different adress in the groundtruth_location'%len(a))
print('%d different adress in the location'%len(b))
```

```
135 different adress in the groundtruth_location
89 different adress in the location
```

De plus, en raison du manque de données dans ce projet, il est impossible d'utiliser des algorithmes ML pour la classification. Par exemple, il manque aussi presque 40 adresses de locations. Cela veut dire qu'il y aura au moins 40 personnes avec une prédiction de location incorrecte. C'est la contrainte.

4.3 Développement futur:

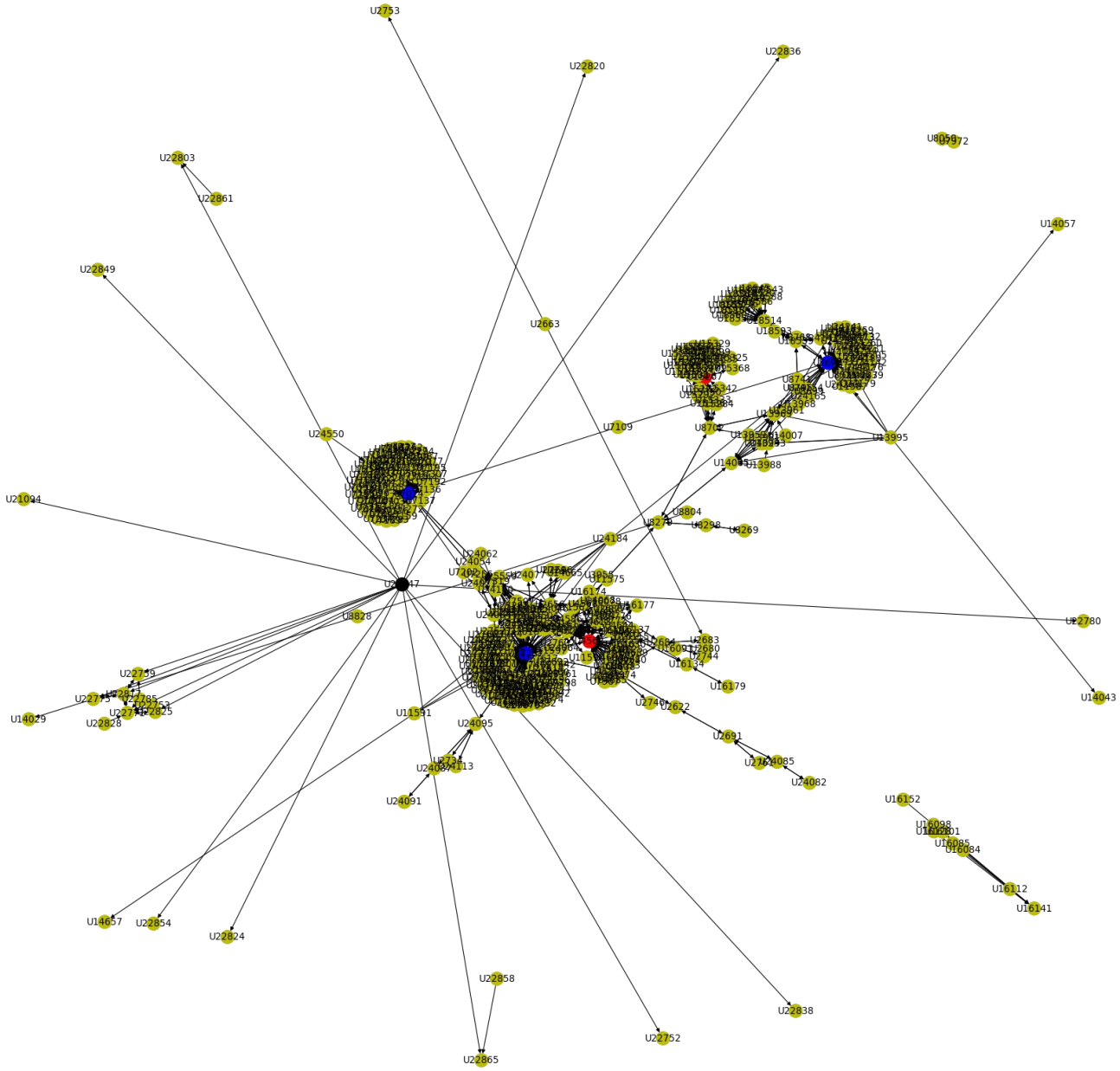
Les attributs peuvent être divisés plus précisément. Par exemple, nous pouvons aussi classifier les noeuds par leur payes (localisation) , par société mère de leur entreprises ou bien par alliance de collège. Ces informations peuvent aussi être extraites à partir des attributs existants.

5. Références：

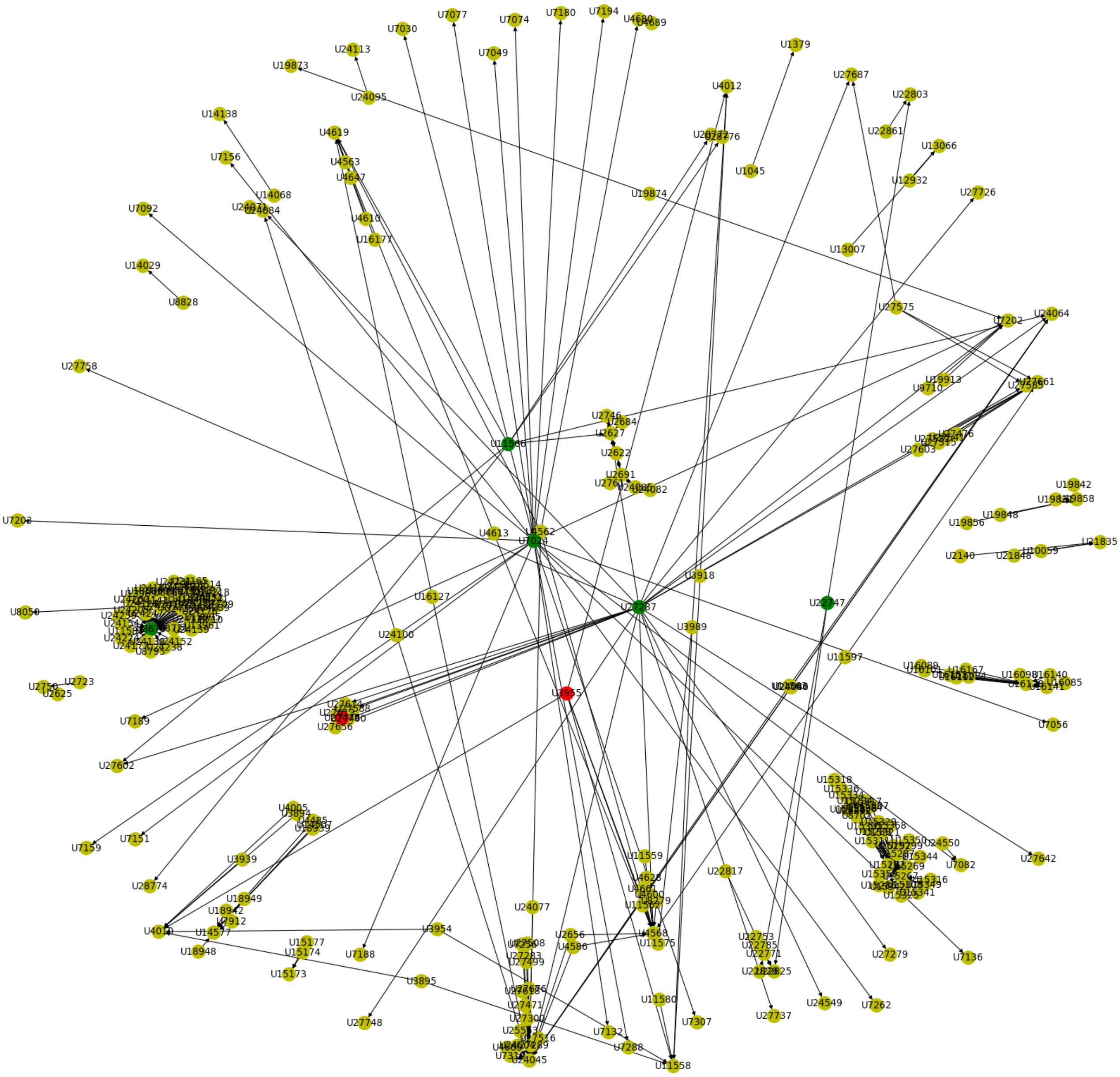
[1] Rui Li,Chi Wang,Kevin Chen-Chuan Chang--User Profilinginan Ego Network:Co-profiling Attributes and Relationships

ANNEXES

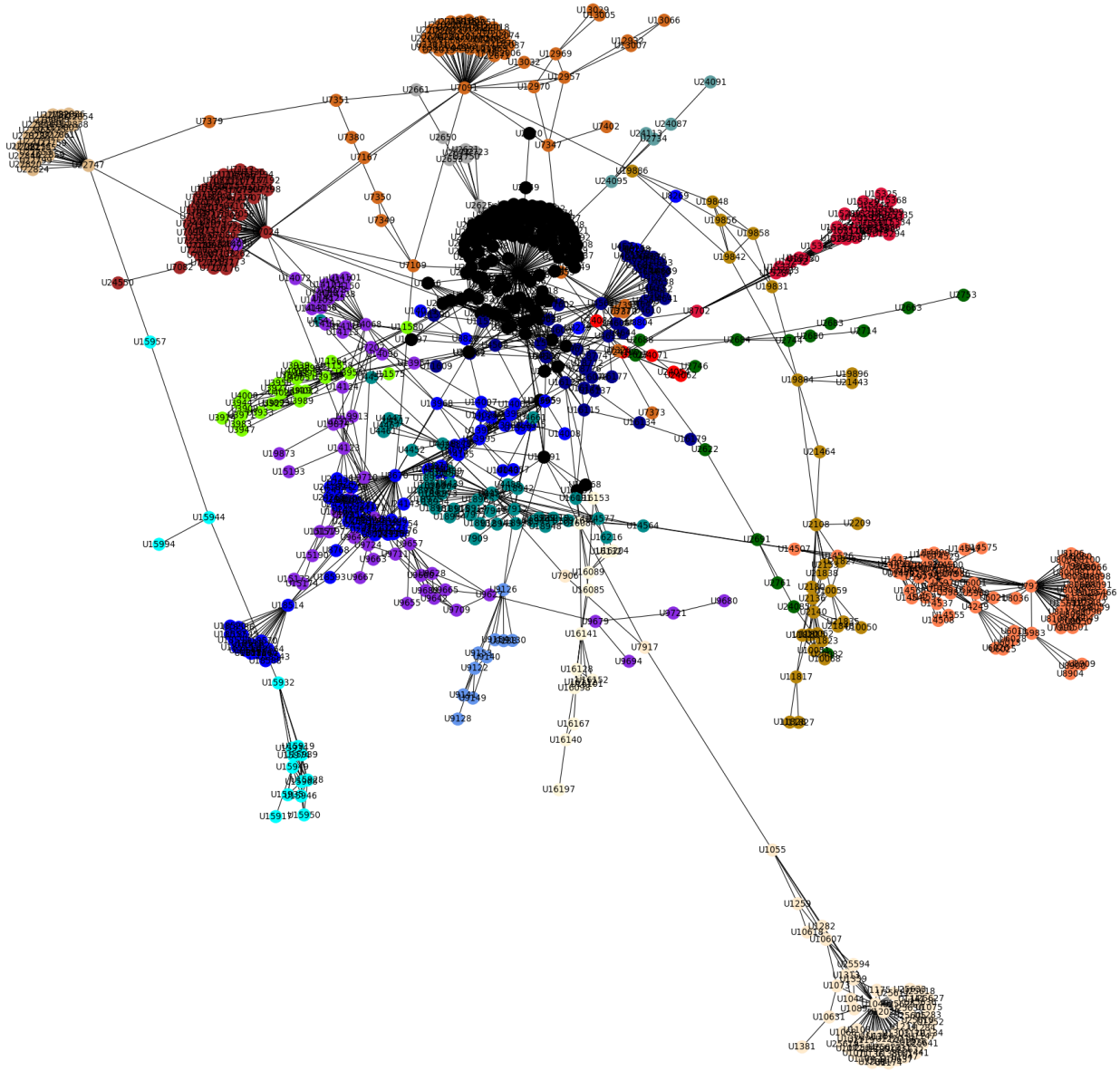
annexe 1 : graphe des 5 influenceurs avec les données prédites



annexe 2 : graphe des 5 influenceurs avec les données vrais


U7030
U7077
U7074
U7180
U7194
U24113
U7049
U1379
U19873
U24095
U4012
U27687
U22803
U14138
U4619
U22861
U13066
U7156
U4563
U4647
U1045
U12932
U14068
U19874
U27726
U7092
U4610
U16177
U13007
U14029
U8828
U27575
U7202
U24064
U27758
U19913
U9710
U2746
U2684
U11566
U2627
U2622
U27603
U2691
U27613
U24082
U19842
U19858
U19848
U19856
U7203
U4613
U4562
U7024
U2140
U21848
U10059
U21835
U3918
U16127
U8050
U22747
U3989
U24100
U11597
U2723
U2750
U2625
U7056
U2955
U7189
U27656
U27602
U15318
U4005
U7159
U7151
U3939
U24550
U7082
U27642
U28774
U11559
U22817
U18949
U24077
U18948
U15177
U15174
U15173
U3954
U7188
U2656
U4586
U11576
U22753
U22785
U22771
U3895
U27279
U7136
U11580
U7307
U27737
U24549
U7262
U27748
U27499
U27283
U27471
U27300
U25513
U7132
U7288
U24045
U11558
U4014
U14577

annexe 3 : Louvain Partition



annexe 4 : Code

Pour la prédiction：

```
import community
from matplotlib import colors as mcolors

def find_community(graph):
	return (community.best_partition(graph))

def prediction_by_community(attr,g=G,empty_nodes=empty_nodes):
	G1 = g
	partition = find_community(G1)
	rst_com=[]
	com_dic = {}
	for index in range(max(partition.values())+1):
	com=[]

	for n in partition:
	if partition[n] == index:
	com.append(n)
	rst_com.append(com)
	com_dic[index] = com

	rst_com = sorted(rst_com, key = lambda t:len(t),reverse = True)
	predictions = {}

	for node in empty_nodes:
	ngb_attr = []
	predictions[node] = []
	for n in com_dic[partition[node]]:
	if n in attr and attr[n]:
	for val in attr[n]:
		ngb_attr.append(val)

	if ngb_attr: # non empty list
	cpt=Counter(ngb_attr)
	a=sorted(cpt.items(), key=lambda t: t[1],reverse = True)
	if len(a)>=2:
	for val in a[:4]:
		predictions[node].append(val[0])
	else:
		predictions[node].append(a[0][0])

	return predictions

def prediction_degree_class(attr = location,G1 = G,empty_nodes = empty_nodes):
	# (i,j) for the degree i ,there are j people having this degree
	degrees = dict(G1.degree())
	node_high_dgr = []
	node_mid_dgr = []
	node_low_dgr = []
	for i in G1:
	if degrees[i] > 10:
	node_high_dgr.append(i)
	elif degrees[i] >= 4 :
	node_mid_dgr.append(i)
```

```
        else:
        node_low_dgr.append(i)
        cpt = Counter(degrees.values())
        cpt = sorted(cpt.items(),key = lambda t: t[0],reverse = False)


        emp = []
        for i in node_high_dgr:
        if i in empty_nodes:
        emp.append(i)
        pre_naiv = prediction_by_community(attr,G1,emp)
        # find_wrong_prediction(pre_naiv,pre2)

        loc_1 = ChainMap(attr,pre_naiv)
        emp = []
        for i in node_mid_dgr:
        if i in empty_nodes:
        emp.append(i)
        pre_naiv1 = prediction_by_community(loc_1,G1,emp)

        loc_2 = ChainMap(loc_1,pre_naiv1)
        emp = []
        for i in node_low_dgr:
        if i in empty_nodes:
        emp.append(i)
        pre_naiv2 = prediction_by_community(loc_2,G1,emp)
        return pre_naiv,pre_naiv1,pre_naiv2
```

Pour la recherche:

```
from collections import ChainMap

pre1,pre2,pre3 = prediction_degree_class()
prediction_location = ChainMap(pre1,pre2,pre3)
print('high degree class')
proba1 = testLocation(pre1)/100
print('\n\nmid degree class')
proba2 = testLocation(pre2)/100
print('\n\nlow degree class')
proba3 = testLocation(pre3)/100
dic1 = ChainMap(location,prediction_location)

emp_noValues = []
for n in G:
        if n not in prediction_location or n not in location:
        emp_noValues.append(n)


# prediction_location_for_novalues = naive_method(dic1,G,emp_noValues)
# prediction_all = ChainMap(prediction_location,prediction_location_for_novalues)

prediction_all = ChainMap(prediction_location)

loc_matched,loc_key_matched = strMatch('bay',location)
loc_matched_pred1,loc_key_matched_pred1 = strMatch('bay',pre1)
loc_matched_pred2,loc_key_matched_pred2 = strMatch('bay',pre2)
loc_matched_pred2,loc_key_matched_pred3 = strMatch('bay',pre3)
```

```
loc_key_matched_pred = loc_key_matched_pred1 + loc_key_matched_pred2 + loc_key_matched_pred3
loc_key_all = loc_key_matched + loc_key_matched_pred
edges = []
G_Bay = nx.DiGraph()

influence_matrix = {}

for i in G:
        influence_matrix[i] = 0
for k in G:
        for ngb in G[k]:
        if ngb in loc_key_matched and ngb not in loc_key_matched_pred:
        edges.append((k,ngb,{'weight':1}))
        influence_matrix[k] += 1
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred1:
        edges.append((k,ngb,{'weight':proba1}))
        influence_matrix[k] += proba1
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred2:
        edges.append((k,ngb,{'weight':proba2}))
        influence_matrix[k] += proba2
        elif ngb not in loc_key_matched and ngb in loc_key_matched_pred3:
        edges.append((k,ngb,{'weight':proba3}))
        influence_matrix[k] += proba3
G_Bay.add_edges_from(edges)

influence_list = [(j,i) for i,j in influence_matrix.items()]
influence_list = sorted(influence_list, reverse = True)

pr = {}
pr1 = {}

for i in G_Bay:
        pr[i] = G_Bay.out_degree(i)

pr_list = [(n_val,n_key) for n_key,n_val in pr.items()]
pr_list = sorted(pr_list,reverse= True)

vals = {}
keys = []
a = 5
for i in range(0,a):
        vals[pr_list[i][1]] = float(pr_list[i][0])
        keys.append(pr_list[i][1])
        G_Bay.node[pr_list[i][1]]['color'] = 'r'
print('\n\n Out_degree top 5')
print(keys)
for i in range(a,len(G_Bay)):
        vals[pr_list[i][1]] = float(pr_list[i][0])
        keys.append(pr_list[i][1])
        G_Bay.node[pr_list[i][1]]['color'] = 'y'

for i in range(a):
        if influence_list[i][1] in G_Bay.nodes() and G_Bay.node[influence_list[i][1]]['color'] == 'r':
        G_Bay.node[influence_list[i][1]]['color'] = 'blue'
        elif influence_list[i][1] in G_Bay.nodes() and G_Bay.node[influence_list[i][1]]['color'] != 'r':
        G_Bay.node[influence_list[i][1]]['color'] = 'black'
        elif influence_list[i][1] not in G_Bay.nodes():
        print(influence_list[i])
```

```
print('influence top 5')
print(influence_list[:10])

drawGraph(G_Bay)
```

Pour dessiner un graphe:

```
def drawGraph(g,color = True):
        print('drawing the Graph, be patient please ^-^')
        plt.figure(num=None, figsize=(30, 30), dpi=80)
        plt.axis('off')
        fig = plt.figure(1)
        pos = nx.spring_layout(g, iterations=100)
        if color == True:
        node_colors = [g.node[v]['color'] for v in g]
        nx.draw_networkx(g,node_color = node_colors)
        elif color == False:
        nx.draw_networkx(g)
        plt.show()
```