Projet de Graph Mining : Diffusion et maximisation d'influence sur les réseaux sociaux

Nicolas Noblot

14 avril 2023

Introduction

- Réseau social = moyen efficace de diffuser de l'information
- Maximisation d'influence = identifier les noeuds d'un réseau qui contribuent le plus à la diffusion de l'information
- Étude de diffusion de l'information et de maximisation d'influence sur Facebook et LastFM Asia
- Plan :
  1. Rappels théoriques
  2. Présentation des datasets
  3. Résultats et discussion

Notations

- Soit $\mathcal{G} = (V, E)$, si $v \in V$, on note $\mathcal{N}(v)$ l'ensemble des voisins de v.
- $\mathcal{S}$: ensemble des graines, noeuds infectés au départ
- $I_t(\mathcal{S})$: ensemble des noeuds infectés à l'instant t dans un processus de diffusion à partir de $\mathcal{S}$
- $I(\mathcal{S}) = \bigcup_{t \geq 0} I_t(\mathcal{S})$: ensemble des noeuds infectés durant un processus de diffusion à partir de $\mathcal{S}$
- $\mathfrak{S}(\mathcal{S}) = \mathbb{E}(|I(\mathcal{S})|)$: espérance du nombre de noeuds infectés à partir de $\mathcal{S}$

Modèle à cascades indépendantes

Algorithme 1 : Algorithme de diffusion par cascades

Entrées : $\mathcal{G} = (V, E)$, $\mathcal{S}$
$I(\mathcal{S}) \leftarrow \mathcal{S}$, $\mathcal{A} \leftarrow \mathcal{S}$
Tant que $\mathcal{A} \neq \varnothing$ **faire**
  $L \leftarrow \varnothing$
  pour chaque $a \in \mathcal{A}$ **faire**
    pour chaque $v \in \mathcal{N}(a)$ **faire**
      Choisir un nombre aléatoire p selon $\mathcal{U}([0,1])$
      si $p < \frac{1}{deg(v)}$ **alors**
        $L \leftarrow L \cup \{v\}$
      sinon
        Ne rien faire
      fin
    fin
  fin
  $\mathcal{A} \leftarrow L \backslash I(\mathcal{S})$
  $I(\mathcal{S}) \leftarrow I(\mathcal{S}) \cup \mathcal{A}$
FinTantque

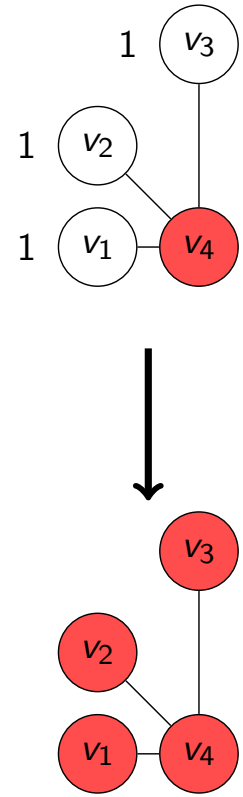


Figure – Exemple de cascade. En rouge, les noeuds infectés . En blanc, les noeuds sains.

Modèle de seuillage

Algorithme 2 : Algorithme de diffusion par seuillage

Entrées : $\mathcal{G} = (V, E)$, $\mathcal{S}$
Initialiser un vecteur θ de seuils aléatoires entre 0 et 1
$I(\mathcal{S}) \leftarrow \mathcal{S}$, $\mathcal{A} \leftarrow \mathcal{S}$
Tant que $\mathcal{A} \neq \varnothing$ **faire**
  $L \leftarrow \varnothing$
  pour chaque $a \in \mathcal{A}$ **faire**
    pour chaque $v \in \mathcal{N}(a)$ **faire**
      Calculer la proportion p de voisins de v infectés
      si $p > \theta_v$ **alors**
        $L \leftarrow L \cup \{v\}$
      sinon
        Ne rien faire
      fin
    fin
  fin
  $\mathcal{A} \leftarrow L \backslash I(\mathcal{S})$
  $I(\mathcal{S}) \leftarrow I(\mathcal{S}) \cup \mathcal{A}$
FinTantque

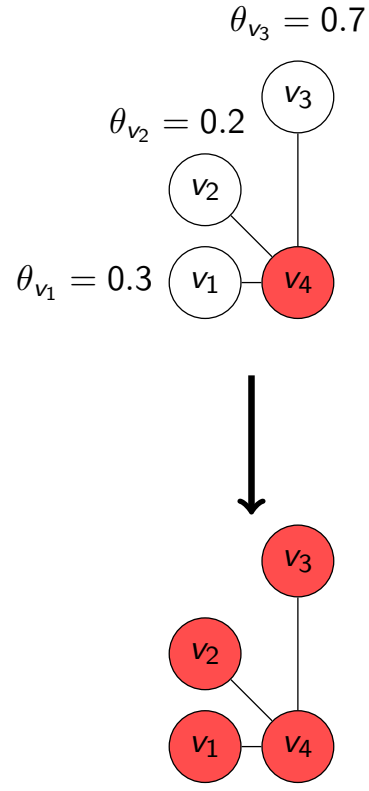


Figure – Exemple d'exécution du modèle de seuillage. Les noeuds infectés sont en rouge.

Heuristique gloutonne de maximisation de l'influence

Algorithme 3 : Heuristique gourmande de maximisation d'influence

Entrées : $\mathcal{G}(V, E)$, *budget* k

$\mathcal{S} \leftarrow \varnothing$

Tant que $|\mathcal{S}| \leq k$ **faire**

  $v^* = \text{argmax}_{v \in V \setminus \mathcal{S}} \left(\mathfrak{S}(\mathcal{S} \cup \{v\})\right) - \mathfrak{S}\mathcal{S}$

  $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$

FinTantque

- Algorithme très calculatoire en pratique.
- Estimation de $\mathfrak{S}(\mathcal{S})$ à l'aide de la moyenne empirique du nombre de noeuds infectés sur plusieurs exécutions d'un algorithme de diffusion.

Dataset ego-Facebook

- Dataset de cercle d'amis sur Facebook
- Les noeuds sont les utilisateurs. Un sommet A est relié à un sommet B si A est ami avec B.
- Graphe non-pondéré et non-orienté
- Statistiques générales :

Statistique	Valeur
Nombre de noeuds	4 039
Nombre d'arêtes	88 234
Densité	0.01
Transitivité	0.52
Coefficient de clustering moyen	0.606
Diamètre	8
Rayon	4

- Source : J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

Dataset LastFM Asia

- Graphe d'utilisateurs en Asie du site LastFM (site de recommandation de musiques)
- Les noeuds sont les utilisateurs et une arête existe entre A et B s'ils se suivent mutuellement.
- Graphe non-pondéré et non-orienté
- Statistiques générales :

Statistique	Valeur
Nombre de noeuds	7 624
Nombre d'arêtes	27 806
Densité	0.00096
Transitivité	0.18
Coefficient de clustering moyen	0.22
Diamètre	15
Rayon	8

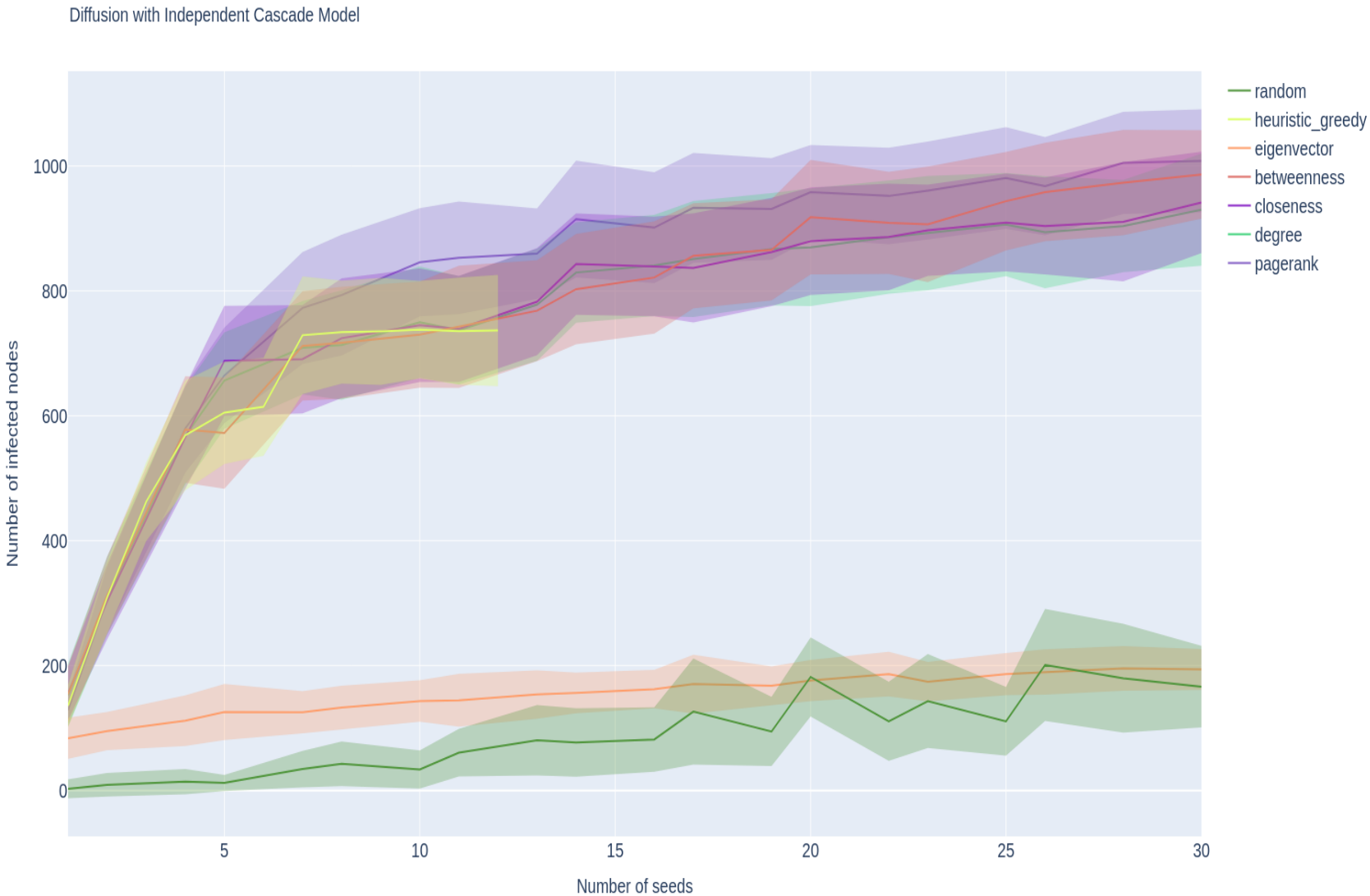
- Source : B. Rozemberczki and R. Sarkar. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. 2020.

Expériences

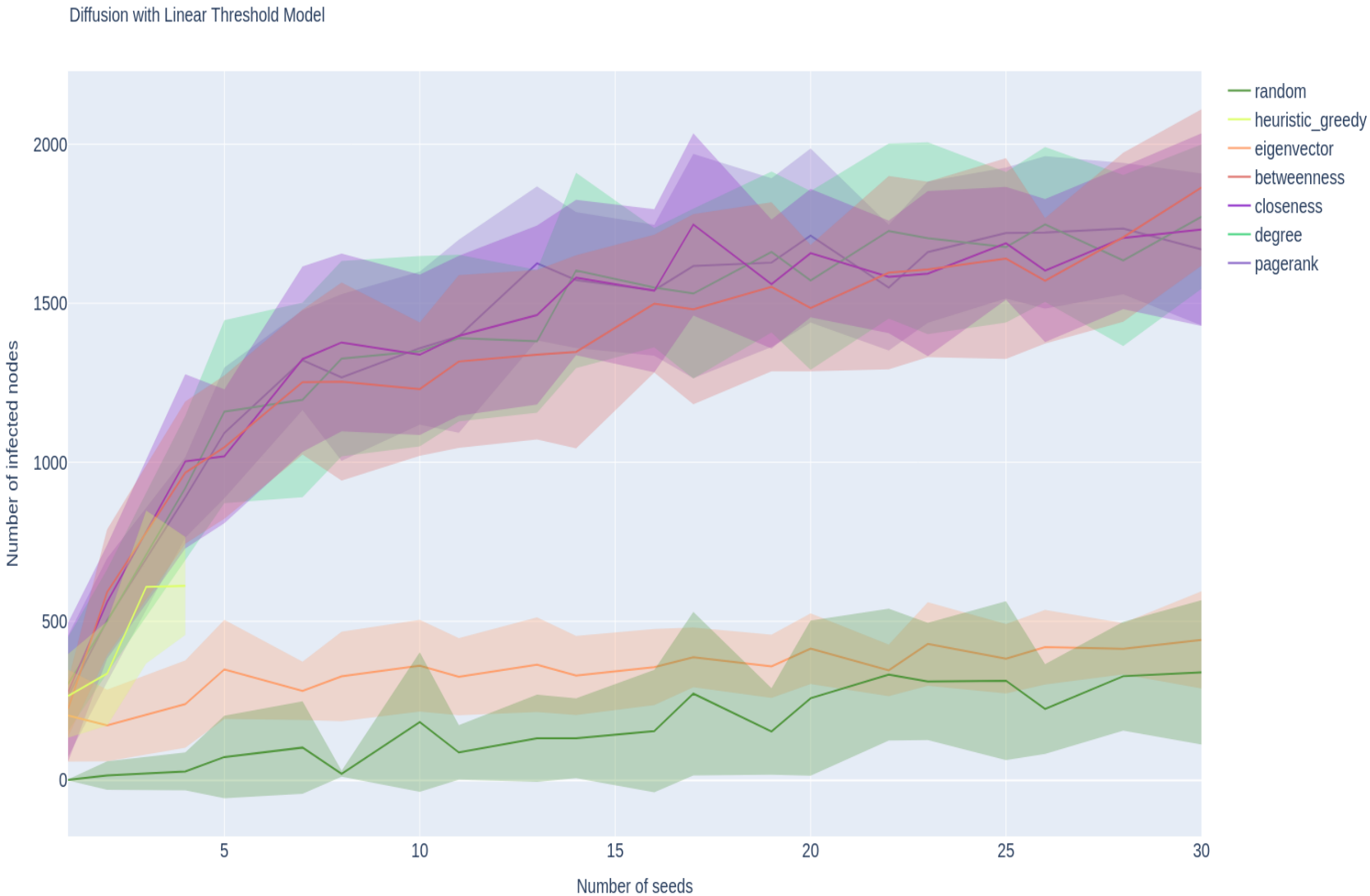
Sur chaque dataset :

- Maximisation d'influence avec un budget inférieur à 30 noeuds
- Utilisation des modèles de seuillage et de cascade avec 6 stratégies de sélection des noeuds de départ différentes : aléatoire, centralité pagerank, centralité closeness, centralité spectrale, centralité intermédiaire, degré
- Comparaison avec l'heuristique gourmande

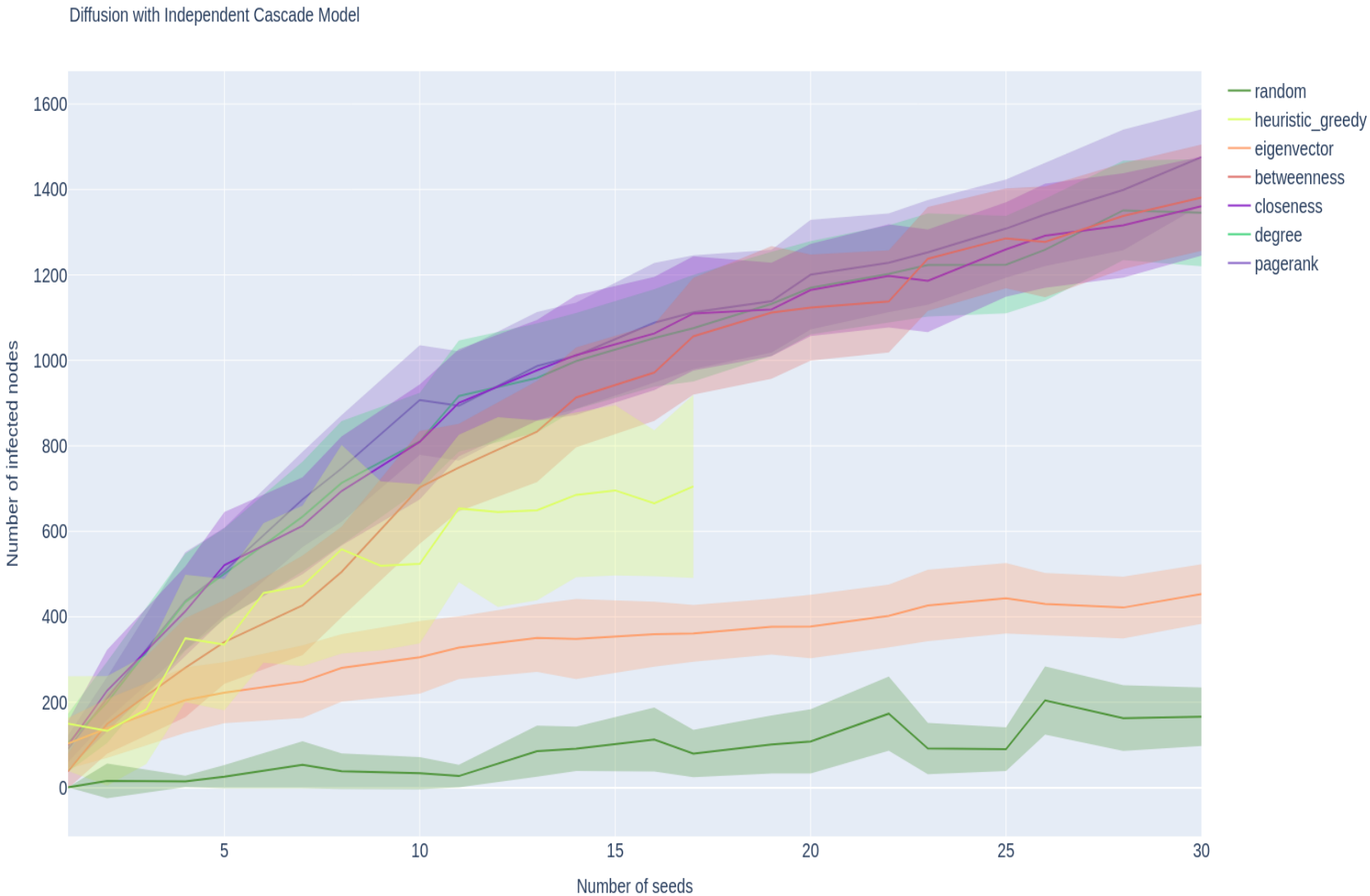
Résultats de la diffusion sur le dataset Facebook (1/2)



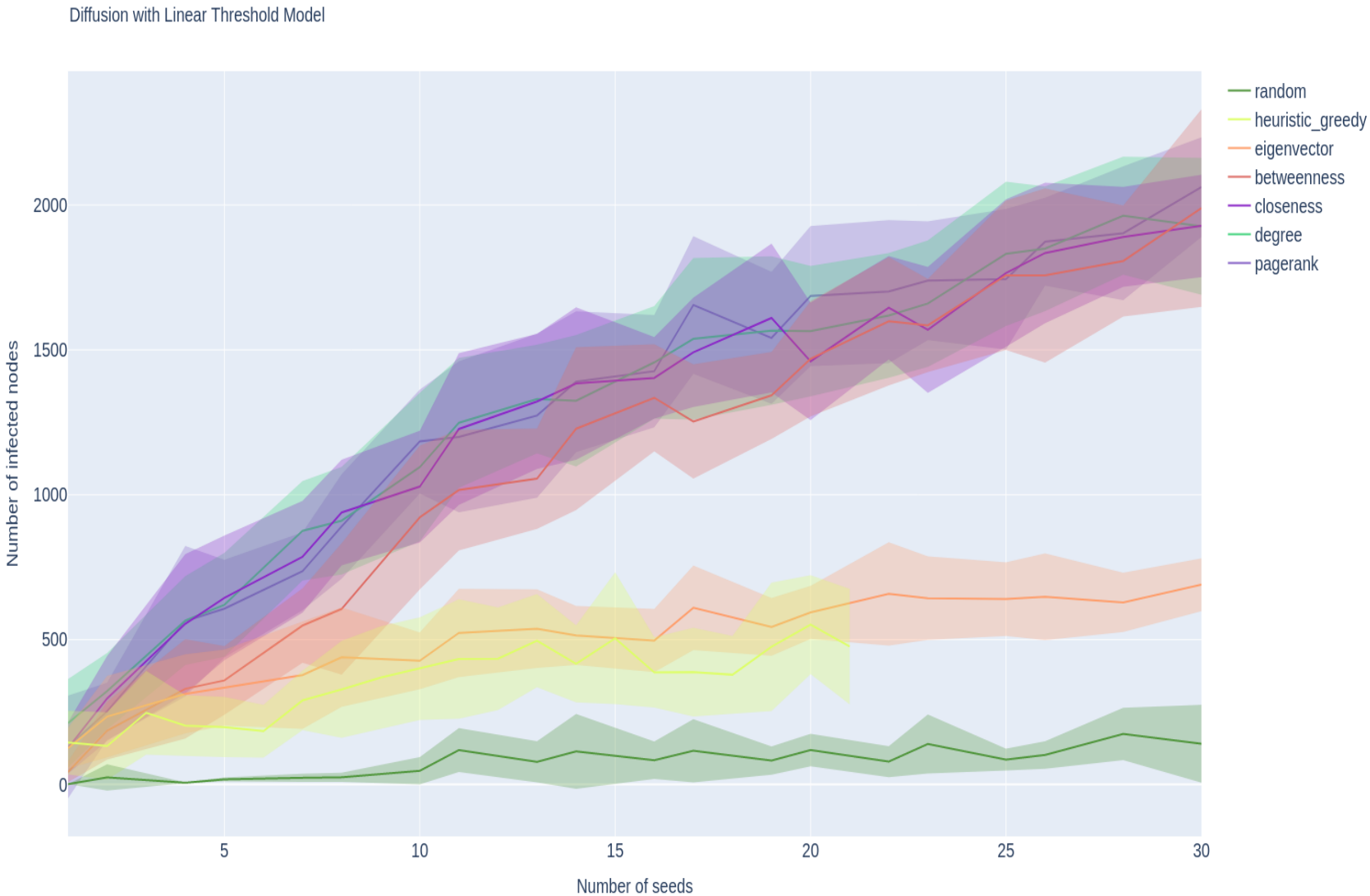
Résultats de la diffusion sur le dataset Facebook (2/2)



Résultats de la diffusion sur le dataset LastFM Asia (1/2)



Résultats de la diffusion sur le dataset LastFM Asia (2/2)



Visualisation des graphes

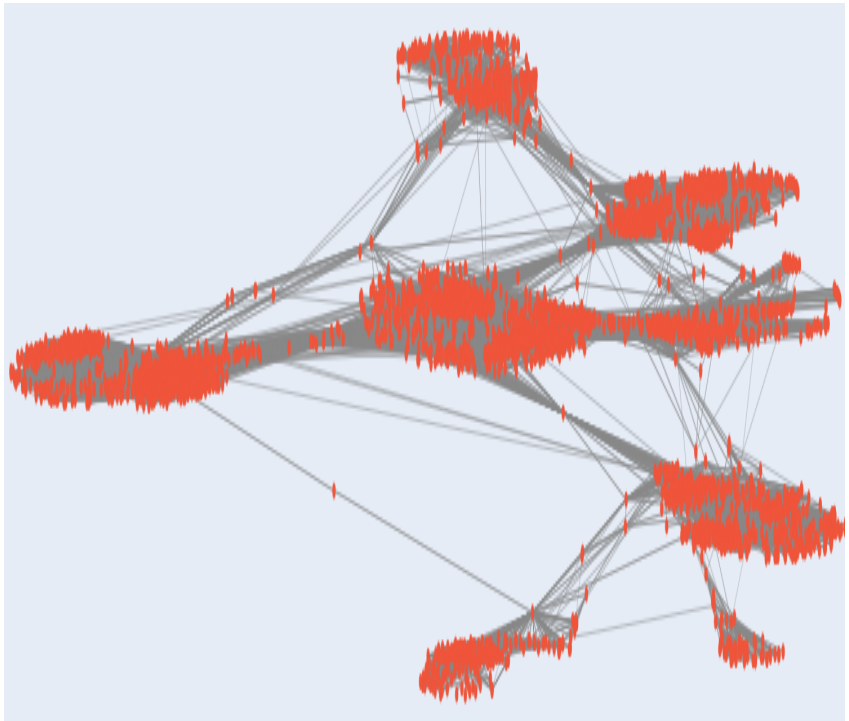


Figure – Projection 2D du graphe de Facebook

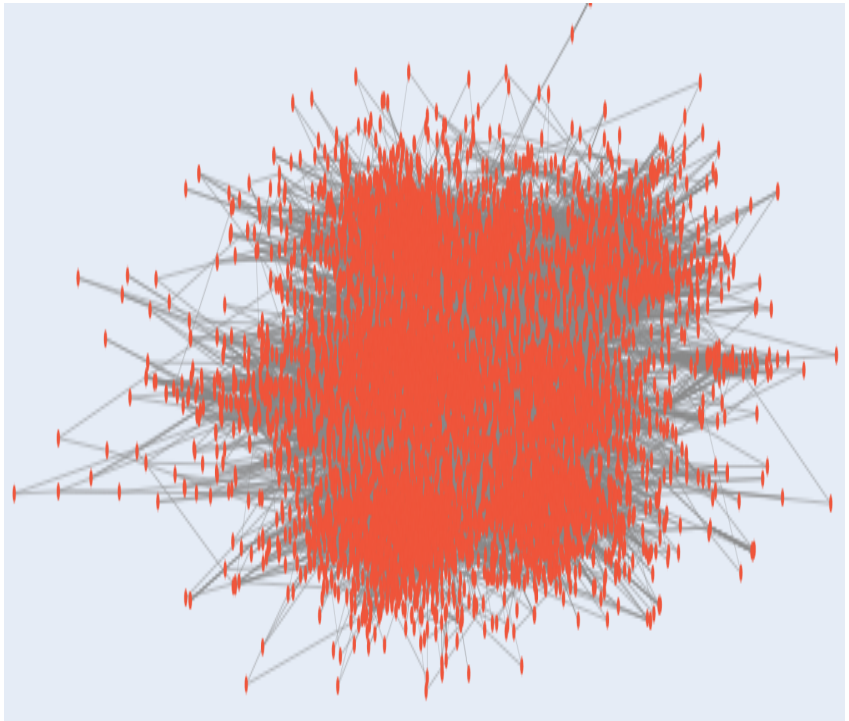


Figure – Projection 2D du graphe de LastFM Asia

Visualisation de centralité

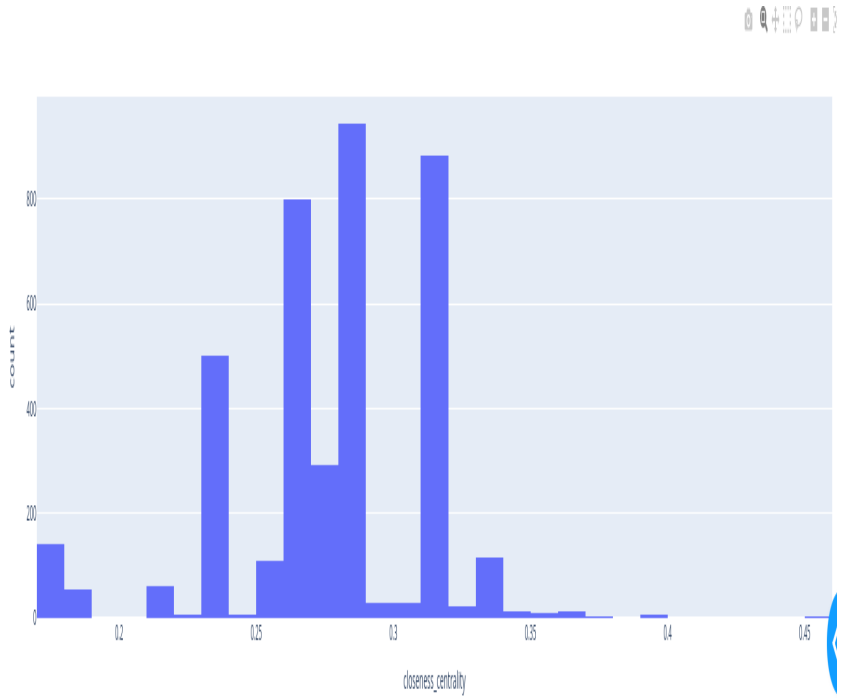


Figure – Distribution de la centralité closeness sur le dataset de Facebook

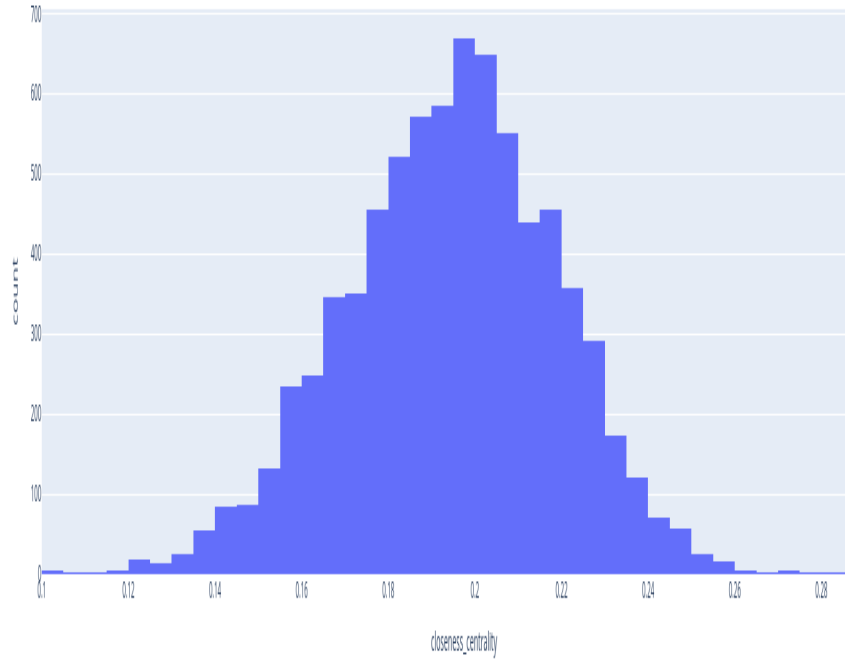


Figure – Distribution de la centralité closeness sur le dataset de LastFM Asia

Sur les deux jeux de données, les autres centralités ont une distribution de type loi exponentielle.

Conclusion

- La diffusion d'information sur les réseaux sociaux dépend de :
  - la stratégie de sélection des germes utilisée
  - du modèle de diffusion
  - de la topologie du réseau
- Algorithmes de diffusion peuvent être coûteux en temps et en ressources même sur des petits réseaux
- Code du projet disponible sur github : `https://github.com/noblotni/graph_mining_labs`


Projet Graph Mining partie Twitch
Etude des méthodes de Label Propagation sur le Twitch Dataset

Présentation du Dataset Twitch_FR

```
Entrée [150]: G.number_of_edges()
   Out[150]: 128983

Entrée [151]: G.number_of_nodes()
   Out[151]: 8824
```

	id	days	mature	views	partner	new_id
0	150417538	577	True	775	False	4867
1	125723704	861	True	2585	False	3692
2	155971814	523	False	1566	False	3816
3	35832890	2234	False	9713	False	416
4	46787750	1896	True	44529	False	4502
...	...	...	...	...	...	...
6546	29196284	2061	False	3822	False	6546
6547	124327806	880	True	30313	False	2392
6548	144229380	640	True	5022	False	391
6549	131423064	736	False	2658	False	2793
6550	45419243	1929	True	3743	False	1092

6551 rows × 6 columns

Tache de classification binaire

Prédire si un utilisateur utilise du langage « mature »

Au niveau du graph, 2 nœuds sont reliés par une arrête si les 2 utilisateur sont mutuellement abonnées l'un à l'autre

Première approche : sans ML

Algorithm 1: $G(V, E)$, labels Y_l

Result: labels $\hat{Y}$

compute $D_{ii} = \sum_j A_{ij}$;

compute $P = D^{-1}A$;

$Y^0 = (Y_l, 0), t = 0$ // Y_u doesn't affect the solution ;

repeat

  $Y^{t+1} \leftarrow PY^t$;

  $Y_l^{t+1} \leftarrow Y_l^t$ //keep the same Y_l;

  $t \leftarrow t + 1$;

until Y^t *converges*;

output Y^t // the most probable label for each node;

Première approche

```
: unlabeled,labeled=train_test_split(target,test_size=0.1)
```

```
Entrée [145]: unlabeled["mature"].value_counts()

  Out[145]: -1    3697
             1    2198
            Name: mature, dtype: int64
```

```
Entrée [146]: labeled["mature"].value_counts()

  Out[146]: -1    438
             1    218
            Name: mature, dtype: int64
```

Résultats de la première approche

```
Entrée [141]: correct/Yl_unlab.shape[0]
   Out[141]: 0.6273112807463953

Entrée [142]: negpred
   Out[142]: 3697

Entrée [143]: pospred
   Out[143]: 1
```

Seconde approche : Iterative classification

```
#G is the complete graph
#train_temp,test_temp=train_test_split(unlabeled)

while unlabeled.shape[0]>0:

    Y_temp=labeled["mature"]
    graph_lab=G.subgraph([n for n in labeled["new_id"]])

    clf = RandomForestClassifier(random_state=0,max_depth=8,n_estimators=100)

    clf.fit(labeled.drop(columns=["mature","new_id"]),Y_temp)

    nodes_with_edge_to_subgraph = [n for n in G.nodes() if n not in graph_lab and any([n in G.neighbors(subgraph_node) for subgr
    new_ids=[int(n) for n in nodes_with_edge_to_subgraph]

    to_label = unlabeled[unlabeled["new_id"].isin(new_ids)]

    y_test=to_label["mature"]

    y_pred=clf.predict(to_label.drop(columns=["mature","new_id"]))
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    to_label["mature"]=y_pred

    print(accuracy,precision,recall,f1)

    unlabeled = unlabeled.drop(to_label.index)
    labeled = pd.concat([labeled, to_label], axis=0)
```

Résultats de la seconde approche

```
accuracy = accuracy_score(target["mature"], labeled["mature"])
precision = precision_score(target["mature"], labeled["mature"], average='weighted')
recall = recall_score(target["mature"], labeled["mature"], average='weighted')
#f1 = f1_score(target["mature"], labeled["mature"], average='weighted')

print("accuracy=",accuracy,"precision=",precision,"recall=",recall)
```

```
accuracy= 0.6640207601892841 precision= 0.6467352823201574 recall= 0.6640207601892841
```

Troisième approche : Choix optimal des nœuds labélisés de départ

Entrée [86]:
```
top_central = target[target["new_id"].isin(sorted_nodes[0:200])]
```

Entrée [75]:
```
top_central
```

Out[75]:

	id	days	mature	views	partner	new_id
16	37799198	2154	-1	117104	-1	3152
28	71914876	1478	1	97104	1	2184
45	43907647	1964	1	4340	-1	308
47	152705416	547	-1	5514	-1	2611
148	184696828	308	1	10328	-1	2476
...	...	...	...	...	...	...
6285	114757037	982	-1	749	-1	1387
6330	31176783	1995	-1	4986	-1	1796
6448	46402303	1909	-1	1815	-1	1102
6476	102520144	1119	1	103397	-1	2168
6530	37444727	2162	-1	2129	-1	2914

197 rows × 6 columns

Entrée [89]:
```
unlabeled,labeled=top_central,target.drop(top_central.index)
```

```
# compute degree centrality for each node
degree_centrality = nx.degree_centrality(G)

# sort nodes by degree centrality
sorted_nodes = sorted(degree_centrality, key=degree_centrality.get, reverse=True)
```

Trosième approche : scoring

```
]: print(labeled["mature"].value_counts())
   print(unlabeled["mature"].value_counts())
```

```
-1    4011
 1    2343
Name: mature, dtype: int64
-1    124
 1     73
Name: mature, dtype: int64
```

```
accuracy = accuracy_score(target["mature"], labeled["mature"])
precision = precision_score(target["mature"], labeled["mature"], average='weighted')
recall = recall_score(target["mature"], labeled["mature"], average='weighted')
#f1 = f1_score(target["mature"], labeled["mature"], average='weighted')

print("accuracy=",accuracy,"precision=",precision,"recall=",recall)
```

```
accuracy= 0.6258586475347275 precision= 0.5960871735052916 recall= 0.6258586475347275
```

Quatrième approche : prédiction d'une autre variable : la variable partner

```
from sklearn.metrics import *

#G is the complete graph
#train_temp,test_temp=train_test_split(unlabeled)

while unlabeled.shape[0]>0:

    Y_temp=labeled["partner"]
    graph_lab=G.subgraph([n for n in labeled["new_id"]])

    clf = RandomForestClassifier(random_state=0,max_depth=8,n_estimators=100)

    clf.fit(labeled.drop(columns=["partner","new_id"]),Y_temp)

    nodes_with_edge_to_subgraph = [n for n in G.nodes() if n not in graph_lab and any([n in G.neighbors(subgraph_node) for subgr
    new_ids=[int(n) for n in nodes_with_edge_to_subgraph]

    to_label = unlabeled[unlabeled["new_id"].isin(new_ids)]

    y_test=to_label["partner"]

    y_pred=clf.predict(to_label.drop(columns=["partner","new_id"]))
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    to_label["partner"]=y_pred

    print("accuracy=",accuracy,"precision=",precision,"recall=",recall)

    unlabeled = unlabeled.drop(to_label.index)
    labeled = pd.concat([labeled, to_label], axis=0)
```

```
accuracy= 0.9705650873603022 precision= 0.9687787150131377 recall= 0.9705650873603022
```