pagerank

May 24, 2022

1 PageRank analysis

In this notebook we've implemented pagerank and compare the results with Networkx implementation. We also run some analytics and interpretation on PageRank results.

[2]:
```
import networkx as nx
import pandas as pd
import numpy as np
import scipy as sp
import seaborn as sns
import matplotlib.pyplot as plt
```

[7]:
```
df_edges = pd.read_csv('../data/csv/edges_deep_link_no_merge.csv')
df_edges['time'] = df_edges['time'] = pd.to_datetime(df_edges['time'], unit='s')
df_edges = df_edges[df_edges['time']> '2022-05-01']
df_edges['time'] = df_edges['time'].dt.date
df_edges.sort_values(by='time', ascending=False)
df_edges = df_edges[df_edges['target'] != '[deleted]']
```

[8]:
```
G = nx.from_pandas_edgelist(df_edges, create_using=nx.MultiDiGraph())
```

[65]:
```
nx.info(G)
```

```
/tmp/ipykernel_85970/1064119803.py:1: DeprecationWarning: info is deprecated and
will be removed in version 3.0.

  nx.info(G)
```

[65]:
```
'MultiDiGraph with 15537 nodes and 58150 edges'
```

[9]:
```
layout = nx.spring_layout(G, seed=500, iterations=100)
```

[10]:
```
nx_ranks = nx.pagerank(G)
```

[11]:
```
def page_rank(G: nx.digraph, iterations=100, alpha=0.85, error=1.0e-6,␣
↪dense=False):
    if len(G)==0: return {}
    nodes = list(G)
```

```
    A = nx.to_numpy_array(G, nodelist=nodes, weight="weight", dtype=float) if␣
↪dense else nx.to_scipy_sparse_array(G, nodelist=nodes, weight="weight",␣
↪dtype=float)
    n, m = A.shape
    if n==0: return {}
    S = A.sum(axis=1)
    S[S != 0] = 1.0 / S[S != 0]
    if dense:
        Q = np.zeros((n,m))
        np.fill_diagonal(Q, S.T.flatten())
    else:
        Q = sp.sparse.csr_array(sp.sparse.spdiags(S.T, 0, n, m))
    A = Q.dot(A)
    x = np.ones(n)/n
    p = np.ones(n)/n
    for _ in range(iterations):
        xlast = x
        x = alpha * (x.dot(A) + sum(x[np.where(S == 0)[0]]) * p) + (1 - alpha)␣
↪* p
        err = np.absolute(x - xlast).sum()
        if err < n * error:
            return dict(zip(nodes, map(float, x)))
    return dict(zip(nodes,map(float,x)))
```

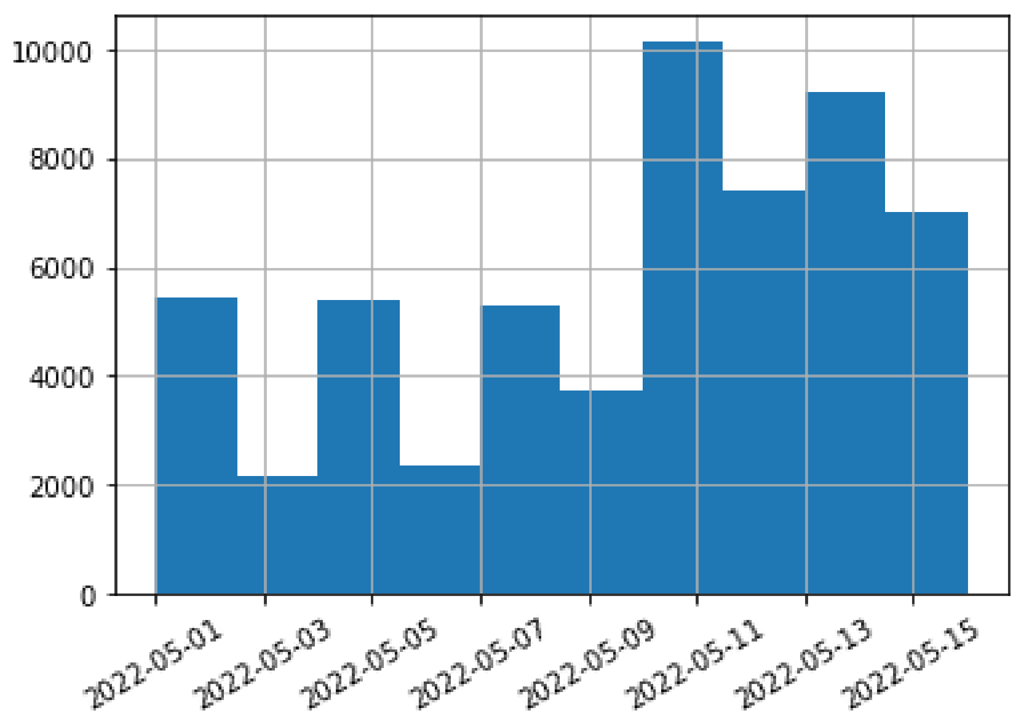
[1]:
```
G = nx.from_pandas_edgelist(pd.read_csv('./data/edges_deep_link_no_merge.csv'),␣
↪create_using=nx.DiGraph())
start = timeit.default_timer()
nx_ranks = nx.pagerank(G)
stop = timeit.default_timer()
nx_time = stop - start
print('networkx pagerank computation time: {}s'.format(nx_time))
nx_memory = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
print('networkx pagerank memory usage: {}MB'.format(nx_memory))
start = timeit.default_timer()
own_ranks = page_rank(G, dense=True)
stop = timeit.default_timer()
own_time = stop - start
own_memory = esource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
print('own pagerank implementation (dense) computation time: {}s'.
↪format(own_time))
print('own pagerank implementation (dense) memory usage: {}MB'.
↪format(own_memory))
start = timeit.default_timer()
own_ranks = page_rank(G, dense=False)
stop = timeit.default_timer()
own_time = stop - start
own_memory = esource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
```

```
print('own pagerank implementation (sparse) computation time: {}s'.
 ↪format(own_time))
print('own pagerank implementation (sparse) memory usage: {}MB'.
 ↪format(own_memory))
overall_error = 3.189112927730331e-16
print(f"overall_error from own implementation to networkx's one:␣
 ↪{overall_error}")
```

```
networkx pagerank computation time: 0.78274936594s
networkx pagerank memory usage: 128.27492749274026MB
own pagerank implementation (dense) computation time: 91.2174837444829s
own pagerank implementation (dense) memory usage: 5812.972137451172MB
own pagerank implementation (sparse) computation time: 2.1811343519406s
own pagerank implementation (sparse) memory usage: 215.154959174MB
overall_error from own implementation to networkx's one: 3.189112927730331e-16
```

Benckmarks above show a real difference between our own implementation and the one from networkx. Yet, this seems to be caused by the dense matrix used. Once using the sparse matrix, it makes a huge difference

[12]:
```
df_edges['time'].hist(bins=10, xrot=30)
plt.savefig('../data/images/comments_repartition.pdf')
```



As it can be seen above, the comments is not exactly uniformly distributed along the days. However, it should not penalise the PageRank interpretation since we are to run the intersections between days.

[13]:
```
df_edges
```

[13]:
```
                 source            target  score  weight        time  \
0            amorydmart     Stevenlerma10     51    51.0  2022-05-15
1               azn1217     Stevenlerma10      9     9.0  2022-05-15
2               azn1217        amorydmart      9     9.0  2022-05-15
3             DanAlucard     Stevenlerma10      9     9.0  2022-05-15
4             DanAlucard        amorydmart      9     9.0  2022-05-15
…                     …                 …     …      …           …
104966      explorer-9  electricmaster23      5     5.0  2022-05-15
104973    geogrant1000  Michellerose6834      3     3.0  2022-05-04
104974    BasicallyTony  Michellerose6834      3     3.0  2022-05-04
104975    BasicallyTony      geogrant1000      3     3.0  2022-05-04
104976          phyLoGG   CRYPTOsauceNews      7     7.0  2022-05-04

                     sub
0               Dogecoin
1               Dogecoin
2               Dogecoin
3               Dogecoin
4               Dogecoin
…                     …
104966           Bitcoin
104973  CryptoCurrencies
104974  CryptoCurrencies
104975  CryptoCurrencies
104976  CryptoCurrencies

[58150 rows x 6 columns]
```

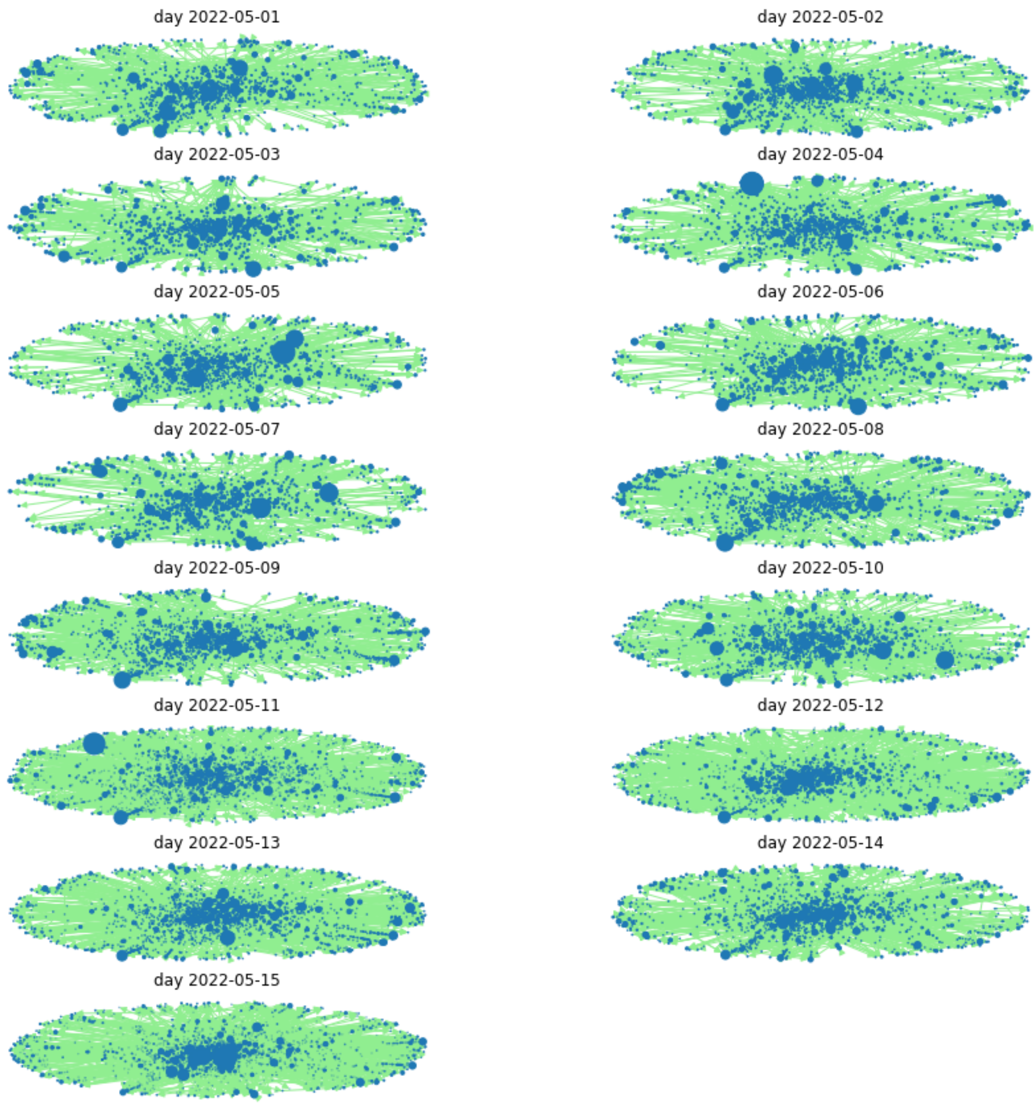
[14]:
```
days = df_edges.groupby('time').groups
days = np.array(list(days.keys()))[:-1]
graphs = [nx.from_pandas_edgelist(df_edges[df_edges['time']==day],␣
 ↪create_using=nx.MultiDiGraph) for day in days]
ranks = [nx.pagerank(g) for g in graphs]
```

[15]:
```
first = graphs[0]
first_rank = ranks[0]
v = np.array(list(first_rank.values()))
first_node_sizes = (v - v.min()) / (v.max() - v.min())
```

[16]:
```
nx.draw(first, pos=layout, node_size=first_node_sizes*300,␣
 ↪edge_color='lightgreen')
```

[57]:
```
rows, columns = len(graphs)//2+1, 2
fig, axs = plt.subplots(rows, columns, figsize=(15,15))
values = []
[[values.append(r[k]) for k in r] for r in ranks]
mn = min(values)
mx = max(values)
axs = axs.flatten()
[ax.axis('off') for ax in axs]
for i, graph in enumerate(graphs):
    r = np.array(list(ranks[i].values()))
    r = (r - mn) / (mx - mn)
    ax=axs[i]
    nx.draw(graph, pos=layout, node_size=r*300, edge_color='lightgreen', ax=ax)
    ax.set_title(f"day {days[i]}")
```

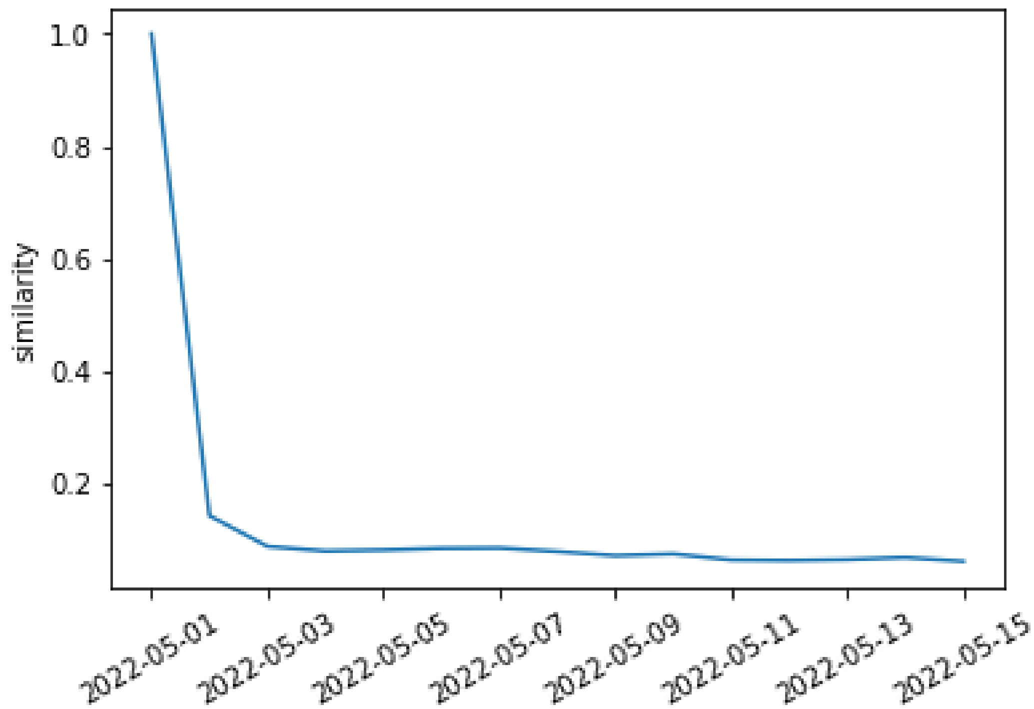
```
9.569495301800634e-05 0.06932999488943387
```



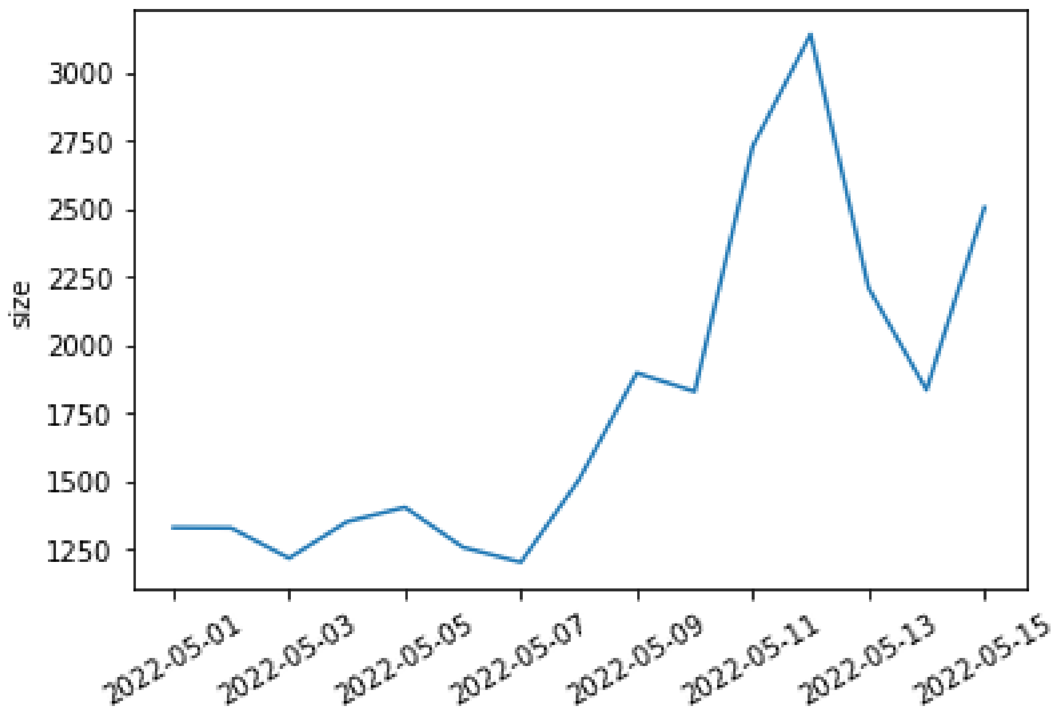
[58]:
```
fig.savefig('../data/images/nx_rep_days.pdf')
```

[59]:
```
error = 0
errors = []
sims = []
prev_rank = ranks[0]
for rank in ranks:
    r = set(rank)
    intersection = set(prev_rank).intersection(r)
    sim = len(intersection) / len(set(prev_rank).union(r))
    sims.append(sim)
    se = sum([(rank[k] - prev_rank[k])**2 for k in intersection])
    errors.append(se)
```

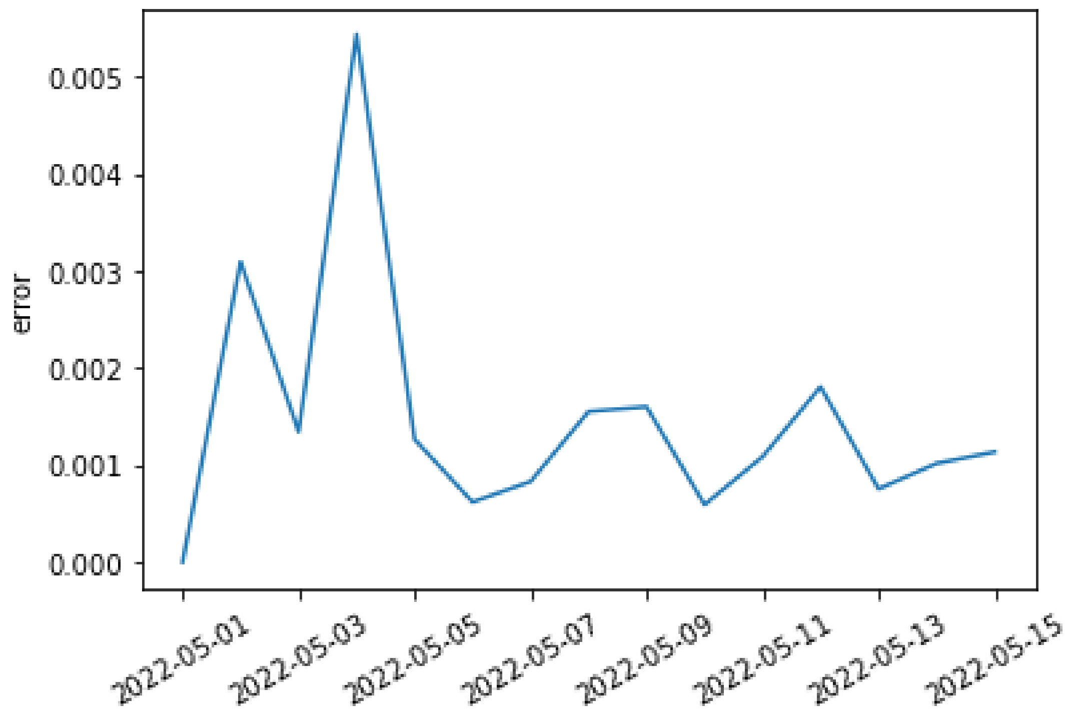
[60]:
```
ax=sns.lineplot(y=sims, x=days)
ax.set(ylabel="similarity")
_=plt.xticks(rotation=30)
plt.savefig('../data/images/sim_days.pdf',bbox_inches='tight')
```



[63]:
```
ax=sns.lineplot(y=[len(r) for r in ranks], x=days)
ax.set(ylabel="size")
_=plt.xticks(rotation=30)
plt.savefig('../data/images/size_days.pdf',bbox_inches='tight')
```



[64]:
```
ax=sns.lineplot(y=errors, x=days)
ax.set(ylabel="error")
_=plt.xticks(rotation=30)
plt.savefig('../data/images/error_days.pdf',bbox_inches='tight')
```



What can be observed above is that although the networks seems to change a lot during the days they remain quite steady in terms of pagerank. The discussion are therefore lasting. This is true that the graphs evolves quite rapidly over the days. The same users won't be active twice on the same posts. Now, there seems to be a high and lasting popularity of users' posts over the days.

```
[ ]:
```