Algorithms

May 24, 2016

Algorithm 1 Weight Assignment algorithm

Input:

1. Data frame $DF(from, to)$, each row signifying an edge between $from$ vertex and to vertex.

Output:

1. Vector wts containing the weight assigned to each edge in DF.

procedure WEIGHT($DF(from, to)$)
    $G(V, E) \leftarrow DF(from, to)$ ▷ $E = (u, v)\ \forall\ u, v \in V$
    $simplify(G)$ ▷ $simplify$ removes all self loops and multiple edges
    $deg \leftarrow degree(G)$
    Let wts be a vector of length $|E|$ initialised to 0
    for each $E_m = (u, v) \in E,\ m = 1, ..., |E|$ **do**
        $start\_c \leftarrow deg[u]$
        $end\_c \leftarrow deg[v]$
        $wts[m] \leftarrow (start\_c + end\_c - 1)/|V| - 1$
    end for
    return wts
end procedure

Algorithm 2 Link Prediction algorithm

Input:

1. Data frame $DF(from, to)$, each row signifying an edge between $from$ vertex and to vertex.
2. Floating-point value $threshold$ for the similarity score.

Output:

1. Data frame $sim(from, to, s)$, each row signifying an edge between $from$ vertex and to vertex with the corresponding similarity score s.

Ensure:

$G(V, E)$ is an incomplete graph.

```
procedure LINK(DF(from, to), threshold)
    G(V, E) ← DF(from, to)                ▷ E = (u, v) ∀ u, v ϵ V
    simplify(G)      ▷ simplify removes all self loops and multiple edges
    fc ← fastgreedy_cluster(G)            ▷ fastgreedy_cluster(G) performs fastgreedy clustering algorithm on graph G and returns each cluster as a subgraph C_i = (V_i, E_i), i = 1, ..., c where c is the number of identified clusters.
    u ← K(V)         ▷ K(V) returns the number of edges required for a complete graph of V vertices
    le ← |E|/u
    lne ← u − |E|/u
    Ω ← le/lne
    for each u, v ϵ V do
        if u ≠ v ∧ (u, v) ∉ E then
            total_cn ← get_common_neighbours(u, v)
            u_cluster ← fc$membership[u]     ▷ fc$membership[u] returns the cluster to which vertex u belongs to
            v_cluster ← fc$membership[v]
            if u_cluster = v_cluster then
                for each w ϵ V do
                    u_cn < −get_common_neighbours(u, w)
                    v_cn < −get_common_neighbours(v, w)
                    if u_cn ≠ 0 ∧ v_cn ≠ 0 then
                        if fc$membership[w] = u_cluster then
                            within_cn + +
                        end if
                    end if
                end for
            else
                within_cn ← 0
            end if
            outside_cn ← within_cn − total_cn
            if within_cn ≠ 0 ∧ outside_cn ≠ 0 then
                s ← within_cn/outside_cn ∗ Ω
            end if
            Let sim be a data frame having three columns, namely, from, to and score initialised to 0
            if s > threshold then
                append_to_data_frame(sim, i, j, s)
            end if
        end if
    end for
    sort_descending_by_column(sim, 3)
    return sim
end procedure
```

Algorithm 2 Link Prediction algorithm (continued)

```
procedure K(G(V, E))
    Let num be an integer variable initialised to 0
    if G is directed then
        num ← |V| * |V − 1|
    else
        num ← |V| * |V − 1|/2
    end if
    return num
end procedure
```

Algorithm 3 Influence Analysis algorithm

Input:

1. Data frame $DF(from, to)$, each row signifying an edge between $from$ vertex and to vertex.
2. Integer $abscut\_off$ which is the cutoff indegree for absolute cut score method.
3. Integer $fixcut\_off$ which is the cutoff percentage for fixed percentage of population method.
4. Integer $sdcut\_off$ which is the number of standard deviations for standard deviation method.
5. Integer num_sim which is the number of simulations to be performed in random permutation method.
6. Floating-point $randcut\_off$ which is the p value for random permutation method.

Output:

1. Vector $abscut$ containing influentials identified by absolute cut score method.
2. Vector $fixcut$ containing influentials identified by fixed percentage of population method.
3. Vector $sdcut$ containing influentials identified by standard deviation method.
4. Vector $randcut$ containing influentials identified by random permutation method.

procedure INFLUENCE($DF(from, to), abscut\_off, fixcut\_off, sdcut\_off, num\_sim, randcut\_off$)
    $G(V, E) \leftarrow DF(from, to)$ ▷ $E = (u, v)\ \forall\ u, v \in V$
    $simplify(G)$ ▷ $simplify$ removes all self loops and multiple edges
    $C \leftarrow FIND\_STABLE\_CLUSTERED\_SAMPLE(G)$ ▷ $FIND\_STABLE\_CLUSTERED\_SAMPLE(G)$ returns a set of clusters identified from a statistically significant sample
    $not\_yet \leftarrow G - C$
    $final \leftarrow CLUSTER\_BY\_PAGERANK(C, not\_yet)$ ▷ $CLUSTER\_BY\_PAGERANK(C, not\_yet)$ returns a set of clusters which have all vertices of not_yet clustered to one of the clusters in C based on pagerank algorithm
    Let $abscut, fixcut, sdcut, randcut$ be vectors initialised to 0.
    for each $cluster(V_{cluster}, E_{cluster}) \in final$ **do**
        $in\_ties \leftarrow in\_degree(cluster)$
        $desc\_in\_ties \leftarrow sort\_decreasing(in\_ties)$
        $scree\_plot \leftarrow plot(desc\_in\_ties)$
        $draw\_horizontal\_abline(scree\_plot, abscut\_off)$
        $append\_points\_above\_abline\_to\_vector(abscut)$
        $cut\_off\_fix \leftarrow fixcut\_off * |V_{cluster}|$
        $draw\_vertical\_abline(scree\_plot, cut\_off\_fix)$
        $append\_points\_left\_of\_abline\_to\_vector(fixcut)$
        $cut\_off\_sd \leftarrow mean(in\_ties) + sdcut\_off * sd(in\_ties)$
        $draw\_horizontal\_abline(scree\_plot, cut\_off\_sd)$
        $append\_points\_above\_abline\_to\_vector(sdcut)$
        $sim \leftarrow simulation\_conditional\_on\_outdegree(cluster, num\_sim)$ ▷ $simulation\_conditional\_on\_outdegree(cluster, num\_sim)$ performs random simulation of $cluster$ graph num_sim number of times constraining on the outdegree.
        Let sim_in_ties be a vector of length $|V_{cluster}| * num\_sim$
        for $j \leftarrow 1\ to\ \#\{sim\}$ **do** ▷ $\#\{sim\}$ returns the number of clusters in sim
            $append\_to\_vector(sim\_in\_ties, indegree(sim[j]))$
        end for

Algorithm 3 Influence Analysis algorithm (continued)

$q \leftarrow quantile(sim\_in\_ties, randcut\_off)$ ▷ $quantile(sim\_in\_ties, randcut\_off)$ returns sample quantiles of sim_in_ties corresponding to the given probability $randcut\_off$
$draw\_histogram(sim\_in\_ties)$
$draw\_vertical\_abline(sim\_in\_ties, q)$
$append\_points\_right\_of\_abline\_to\_vector(randcut)$
end for
return $abscut, fixcut, sdcut, randcut$
end procedure

procedure FIND_STABLE_CLUSTERED_SAMPLE($G(V, E)$)
$perc \leftarrow 0.1$
Let $major\_fc\_length$, $temp$ and $diff$ be vectors of length 10 initialised to 0.
$i \leftarrow 1$
while $perc \leqslant 1$ **do**
$len \leftarrow |V| * perc$
$sample \leftarrow random\_sample\_without\_repetition(G, len)$ ▷ $random\_sample\_without\_repetition(G, len)$ randomly selects len number of vertices and edges corresponding to these vertices from graph G
$fc \leftarrow fastgreedy\_cluster(sample)$ ▷ $fastgreedy\_cluster(sample)$ performs fastgreedy clustering algorithm on graph $sample$ and returns each cluster as a subgraph $C_i = (V_i, E_i), i = 1, ..., c$ where c is the number of identified clusters.
$major\_fc \leftarrow fc[sizes(fc) > (0.01 * len)]$ ▷ $sizes(fc)$ returns the number of vertices in each cluster present in fc
$major\_fc\_length[i] \leftarrow \#\{major\_fc\}$
$temp[i] \leftarrow major\_fc\_length[i]$
$res \leftarrow 0$
if $i > 1$ **then**
$diff[i-1] \leftarrow |temp[i] - temp[i-1]|$
if $i > 2$ **then**
if $|diff[i-1] - diff[i-2]| \leqslant res$ **then**
$res \leftarrow |diff[i-1] - diff[i-2]|$
break
else
$res \leftarrow |diff[i-1] - diff[i-2]|$
end if
end if
end if
$i++$
$perc \leftarrow perc + 0.1$
end while
return $major\_fc$
end procedure

Algorithm 3 Influence Analysis algorithm (continued)

procedure CLUSTER_BY_PAGERANK($C(V_c, E_c), not\_yet(V_c', E_c')$)
  Let *pageranks* be vector of length $\#\{C\}$ initialised to 0.
  while $\#|V_{c'}| \neq 0$ **do**
    $random(key, E_{key}) \leftarrow randomly\_select\_subgraph\_without\_repetition(not\_yet, 1)$ ▻ $randomly\_select\_subgraph\_without\_repetition(not\_yet, 1)$ randomly selects 1 vertex, *key* from *not_yet* and corresponding edges $E_{key} = (key, V_o)$, V_o being vertices in not_yet having edge to *key*.
    $delete\_from\_graph(not\_yet, random)$
    for each $C_i = (V_i, E_i) \in C,\ i = 1, ..., \#\{C\}$ **do**
      Let $check(V_{check}, E_{check})$ be an empty graph object.
      $check \leftarrow C_i$
      $append\_to\_list(V_{check}, key)$
      for each $E_{jkey} = (key, V_{jo}) \in E_{key},\ j = 1, ..., |E_{key}|$ **do**
        if $V_{jo} \in V_{check}$ **then**
          $append\_to\_list(E_{check}, E_{jkey})$
        end if
      end for
      $pr \leftarrow pagerank(check)[key]$
      $pageranks[i] \leftarrow pr$
    end for
    if $sum(pageranks) > 0$ **then**
      $ind \leftarrow index(max(pageranks))$
      $append\_to\_list(V_{ind}, key)$
      for each $E_{jkey} = (key, V_{jo}) \in E_{key},\ j = 1, ..., |E_{key}|$ **do**
        if $V_{jo} \in V_{ind}$ **then**
          $append\_to\_list(E_{ind}, E_{jkey})$
        end if
      end for
    end if
  end while
  return C
end procedure

Algorithm 4 Time Series Analysis algorithm

Input:

1. Data frame $DF(from, to, time)$, each row signifying an edge between $from$ vertex and to vertex with the corresponding time, $time$
2. Floating-point value $start$ indicating the value of $time$ when analysis must commence.
3. Floating-point value end indicating the value of $time$ when analysis must terminate.
4. Floating-point value $increment$ indicating the increments of $start$ in every iteration.

Output:

1. Set of plots, $images(plots)$ containing the plots of the graph at every iteration.

procedure TIME($DF(from, to, time), start, end, increment$)
    $G(V, E, T) \leftarrow DF(from, to, time)$ ▷ $E = (u, v, t)\ \forall\ u, v \in V,\ \forall\ t \in T$
    $vcolor \leftarrow generate\_color\_palette(V)$ ▷ $generate\_color\_palette(V)$ returns a distinct color for each vertex in V
    $i \leftarrow start$
    while $i \leqslant end$ **do**
        for each $e \in E,\ v \in V$ **do**
            if $e\$time < i$ **then**
                $e\$weight \leftarrow 1$
                $e\$color \leftarrow "gray"$
            else
                $e\$weight \leftarrow 0$
                $e\$color \leftarrow "black"$
            end if
            if $strength(v) = 0$ **then** ▷ $strength$ returns the sum of edge weights of adjacent edges of vertex v
                $v\$color \leftarrow "black"$
            else
                $v\$color \leftarrow vcolor[v]$
            end if
            $v\$size \leftarrow 1 + 2 * log(strength(v))$
        end for
        $append\_to\_plots(images, plots)$
        $i \leftarrow i + increment$
    end while
end procedure