Rotinas para recomendação de recursos

Estas rotinas são acionadas pelo sistema de recomendação de recursos e dependem das estruturas auxiliares.

In [72]:
```
from SPARQLWrapper import SPARQLWrapper, JSON
import networkx as x, nltk as k, numpy as n
import string, time, random, __builtin__
```

In [13]:
```
URL_ENDPOINT="http://localhost:82/"
URL_ENDPOINT_=URL_ENDPOINT+"participabr/query"
EXCLUDE=set(string.punctuation+u'\u201c'+u'\u2018'+u'\u201d'+u'\u2022'+u'\
STOPWORDS=set(k.corpus.stopwords.words('portuguese'))
PREFIX="""PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ops: <http://purl.org/socialparticipation/ops#>
PREFIX opa: <http://purl.org/socialparticipation/opa#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX tsioc: <http://rdfs.org/sioc/types#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX schema: <http://schema.org/>"""
stemmer = k.stem.RSLPStemmer()
```

carregando as estruturas auxiliares:

In [45]:
```
g=pickle.load( open( "pickledir/g.p", "rb" ) )
d=pickle.load( open( "pickledir/d.p", "rb" ) )
bow=pickle.load( open( "pickledir/bow.p", "rb" ) )
radicais_escolhidos=pickle.load( open( "pickledir/radicais_escolhidos.p",
bows=pickle.load( open( "pickledir/bows.p", "rb" ) )
```

realizando estruturas derivadas:

In [19]:
```
d_=x.Graph()
efoo=d.edges(data=True)
for e in efoo:
    if d_.has_edge(e[0],e[1]):
        d_[e[0]][e[1]]["weight"]+=e[2]["weight"]
    else:
        d_.add_edge(e[0],e[1],weight=e[2]["weight"])
eg=g.edges()
ed=d.edges(data=True)
ed_=d_.edges(data=True)
```

função auxiliar:

In [21]:
```
def NL(narray):
    return narray/narray.sum()
```

Recomendação de participantes:

recomendaParticipante(destinatario, idd=0,
metodo="topologico",polaridade="ambas",ordenacao="compartimentada"):

In [89]:
```
#destinatario="linha_editorial"
destinatario="participante"
idd="rfabbri"
metodo="hibrido" # topologico+textual
polaridade="ambas" # similar e dissimilar
ordenacao="compartimentada"
recomendacoes=[]
```

Recomendação de participantes para linha editorial:

In [90]:
```
if destinatario=="linha_editorial":
    if metodo in ("topologico","hibrido"):
        ###
        # topologico
        # puxa a rede em si, retorna geral
        wd=d.degree(weight="weight")
        wd_=[(i,wd[i]) for i in wd.keys()]
        wd_.sort(key=lambda x: -x[1])
        recomendados=[i[0] for i in wd_]
        pontuacao=[i[1] for i in wd_]
        criterio="numero de interacoes (forca do participante no grafo de
        recomendacoes.append({"recomendados":recomendados,
                              "pontuacao":pontuacao,
                              "criterio":criterio})
        ud=d.degree()
        ud_=[(i,ud[i]) for i in ud.keys()]
        ud_.sort(key=lambda x: -x[1])
        recomendados=[i[0] for i in ud_]
        pontuacao=[i[1] for i in ud_]
        criterio="quantidade de participantes com que interagiu (grau do p
        recomendacoes.append({"recomendados":recomendados,
                              "pontuacao":pontuacao,
                              "criterio":criterio})
        gd=g.degree()
        gd_=[(i,gd[i]) for i in gd.keys()]
        gd_.sort(key=lambda x: -x[1])
        recomendados=[i[0] for i in gd_]
        pontuacao=[i[1] for i in gd_]
        criterio="quantidade de participantes amigos"
        recomendacoes.append({"recomendados":recomendados,
                              "pontuacao":pontuacao,
                              "criterio":criterio})

    if metodo in ("textual","hibrido"):
        ###
        # textual
        # usa BoW para comparar os usuarios com a media geral,
        # retorna dos mais típicos e os outliers
        ocorrencias=[bow[i] for i in radicais_escolhidos]
        bow=n.array(ocorrencias,dtype=float)
        bowNL=NL(bow)
        rec=[]
```

```
        for uri_ in bows.keys():
            bow_=n.array(bows[uri_][1],dtype=n.float)
            if bow_.sum():
                distancia=n.sum(bowNL-NL(bow_))**2
                rec.append((uri_,distancia))
        rec.sort(key = lambda x: x[1])
        recomendados=[i[0] for i in rec]
        pontuacao=[1/(i[1]+1) for i in rec]
        criterio="semelhanca com o vocabulario do participa.br como um tod
        recomendacoes.append({"recomendados":recomendados,
                  "pontuacao":pontuacao,
                  "criterio":criterio})
```

Recomendação de participante para participante

In [91]:

```
if destinatario=="participante":
    uri="http://participa.br/profile/"+idd
    if metodo in ("topologico","hibrido"):
        ###
        # todos os participantes x_n com que interagiu,
        # na ordem decrescente de interação:
        # {d_i}_0^n, I[x_n]>=I[x_(n-1)],
        # com I a intensidade interacao, o número de mensagens trocadas
        if uri in d_.nodes():
            x_n=d_[uri]
            x_n_=[(i,x_n[i]["weight"]) for i in x_n.keys()]
            x_n_.sort(key=lambda x: -x[1])

            # é feita sugestão dos participantes que não são amigos:
            # x_n!=g_n, g_n um amigo
            if uri in g.nodes():
                viz=g.neighbors(uri)
                x_n_=[i for i in x_n_ if i[0] not in viz]
            recomendados=[i[0] for i in x_n_]
            pontuacao=[i[1] for i in x_n_]
            criterio="numero de interacoes"
            recomendacoes.append({"recomendados":recomendados,
                                  "pontuacao":pontuacao,
                                  "criterio":criterio})
        ###
        # avançado e talvez desnecessário: recomenda usuários
        # com quem os amigos mais interagiram
        # e q jah n sao amigos do participante que recebe a recomendação
        # pode ficar pesado quando o usuário tiver muitos amigos

        ###
        # achar amigo de amigo, excluir amigos e recomendar
        if uri in g.nodes():
            vizs=g.neighbors(uri)
            vizs_=set(vizs)
            vv=[]
            for viz in vizs:
                vv+=g.neighbors(viz)
            vv_=list(set(vv))
            candidatos=[(i,vv.count(i)) for i in vv_ if i not in vizs_]
```

```
            candidatos.sort(key=lambda x: -x[1])

            recomendados=[i[0] for i in candidatos]
            pontuacao=[i[1] for i in candidatos]
            criterio="mais amigos em comum"
            recomendacoes.append({"recomendados":recomendados,
                                  "pontuacao":pontuacao,
                                  "criterio":criterio})
        ###
        if polaridade in ("dissimilar","ambas"):
            recomendacoesD=[] # para recomendacoes com polaridade dissimil
            ### maiores geodesicas partindo do destinatario.
            if uri in g.nodes():
                caminhos=x.shortest_paths.single_source_shortest_path(g,ur
                caminhos_=[caminhos[i] for i in caminhos.keys()]
                caminhos_.sort(key=lambda x : -len(x))
                #distantes=[(i[-1],len(i)) for i in caminhos_]
                recomendados=[i[-1] for i in caminhos_ if len(i)>2]
                pontuacao=  [len(i) for i in caminhos_ if len(i)>2]
                criterio="participantes na mesma rede de amizades, mas mai
                recomendacoesD.append({"recomendados": recomendados,
                                       "pontuacao":pontuacao,
                                       "criterio":criterio})
            # feito para amigos, agora com a rede de interacao
            if uri in d.nodes():
                caminhos=x.shortest_paths.single_source_shortest_path(d,ur
                caminhos_=[caminhos[i] for i in caminhos.keys()]
                caminhos_.sort(key=lambda x : -len(x))
                recomendados=[i[-1] for i in caminhos_ if len(i)>2]
                pontuacao=  [len(i) for i in caminhos_ if len(i)>2]
                criterio="participantes na mesma rede de amizades, mas mai
                recomendacoesD.append({"recomendados":recomendados,
                                       "pontuacao":pontuacao,
                                       "criterio":criterio})
            # participantes de outras componentes conexas com relacao ao d
            if uri in g.nodes():
                comps=x.connected_components(g)
                # caso haja duas componentes conexas
                if len(comps)>1:
                    recomendados=[]
                    # caso sejam exatamente duas componentes:
                    if len(comps)==2:
                       for comp in comps:
                            if uri not in comp:
                                # recomenda a componente toda
                                recomendados+=[(i,1) for i in comp]
                                criterio="participantes da unica component
                    # caso sejam mais de duas componentes:
                    else:
                        for comp in comps:
                            if uri not in comp:
                                # escolhe participante da componente
                                recomendados.append((random.sample(comp,1)
                                criterio="participante de componente de am
                    recomendados_=[i[0] for i in recomendados]
                    pontuacao=[i[1] for i in recomendados]
                    recomendacoesD.append({"recomendados": recomendados_,
                                       "pontuacao":pontuacao,
                                       "criterio":criterio})
    if metodo in ("textual","hibrido"):
        # acha amigos
```

```
        if uri in g.nodes():
            amigos=g.neighbors(uri)
        else:
            amigos=[]
        # verifica se bow eh vazia
        # listar pelos que tem vocabulário mais semelhante
        # segundo critério de menor distancia euclidiana
        bowi=bows[uri]
        if bowi[0].N() == 0:
            # bow do destinatario vazia, usando media geral:
            ocorrencias=[bow[i] for i in radicais_escolhidos]
            bowi=n.array(ocorrencias,dtype=float)
        else:
            bowi=n.array(bowi[1],dtype=float)
        uris=bows.keys()
        rec=[]
        for uri_ in uris:
            if (uri_ != uri) and (uri_ not in amigos):
                bowj=n.array(bows[uri_][1],dtype=n.float)
                distancia=n.sum(NL(bowi)-NL(bowj))**2
                rec.append((uri_,distancia))
        rec.sort(key = lambda x: x[1])
        if len(rec)>0:
            recomendados=[i[0] for i in rec]
            pontuacao=[1/(i[1]+1) for i in rec]
            criterio="semelhanca dentre vocabularios E (0,1]. Calculo: sem
            recomendacoes.append({"recomendados":recomendados,
                      "pontuacao":pontuacao,
                      "criterio":criterio})
    if metodo=="hibrido":
        # fazer medida composta de vocabulario e proximidade na rede de in
        # fazer medida composta de vocabulario e proximidade na rede de am
        # pega amigo de amigo, rankeia por media de amigos em comum e voca
    ## polaridade negativa:
        # pega amigo de amigo, rankeia por inverso da media de amigos em c
        pass
```

In [92]:
```
len(recomendacoes)
```

Out[92]: 3

Ajustando polaridade da recomendação

Se de similaridade, dissimilaridade ou ambas

In [93]:
```
if polaridade in ("dissimilar","ambas"):
    try:
        recomendacoesD
    except:
        recomendacoesD=[]
```

```
    # inversao das ordenacoes anteriores
    for i in xrange(len(recomendacoes)):
        recomendacoesD.append({})
        recomendacoesD[i]["recomendados"]=recomendacoes[i]["recomendados"]
        recomendacoesD[i]["pontuacao"]=recomendacoes[i]["pontuacao"][::-1]
        recomendacoesD[i]["criterio"]="INVERTIDO: "+recomendacoes[i]["crit
    if polaridade == "ambas":
        recomendacoes=recomendacoes+recomendacoesD
    else:
        recomendacoes=recomendacoesD
```

Formato de entrega das recomendações

Se embaralhada ou intercalada

In [94]:
```
# o embaralhamento e intercalação são cortezias da api
if ordenacao=="embaralhada":
    recs=[]
    for i in xrange(len(recomendacoes)):
        tanto=int(len(recomendacoes[i]["recomendados"])*0.1)
        if tanto < 2:
            tanto=min(5,len(recomendacoes[i]["recomendados"]))
        for j in xrange(tanto):
            recomendado=recomendacoes[i]["recomendados"][j]
            pontuacao=recomendacoes[i]["pontuacao"][j]
            criterio=recomendacoes[i]["criterio"]
            recs.append((recomendado, pontuacao, criterio))
    random.shuffle(recs)
    recomendacoes=recs
if ordenacao=="intercalada":
    recs=[]
    cond=1
    cont=0
    while cond:
        for i in xrange(len(recomendacoes)):
            if cont<len(recomendacoes[i]["recomendados"]):
                recomendado=recomendacoes[i]["recomendados"][cont]
                pontuacao=recomendacoes[i]["pontuacao"][cont]
                criterio=recomendacoes[i]["criterio"]
                recs.append((recomendado, pontuacao, criterio))
        cont+=1
        if cont>=7:
            cond=0
    recomendacoes=recs
```

In [95]:
```
len(recomendacoes)
```

Out[95]: 9

Recomendação de demais recursos

In [101]:
```
def recomendaComunidade(destinatario, idd, metodo="hibrido",polaridade="am
```

```
    # recomenda por vocabulario em comum do usado na comunidade com o part
    #### puxar dados:
    # texto produzido pela comunidade
    # participantes na comunidade

    # por possuir membros amigos ou que interagiram muito
    # por possuir mais amigos de amigos

    # semelhanca de vocabulario

    # mais amigos de pessoas com quem interagiu
    # mais pessoas que iteragiram com seus amigos
    # por media de amigos e vocabulario utilizado
    pass
def recomendaTrilha(destinatario, idd, metodo="hibrido", polaridade="mista
    # que prazo final nao tenha passado
    # e prazo inicial esteja proximo
    # que possui amigos que colaboraram
    # que possui amigos e pessoas que interagiram com o destinatario
    # cujos textos sao proximos aos do participante
    pass
def recomendaArtigo(destinatario, idd, metodo="hibrido", polaridade="mista
    if destinatario=="participante":
        uri="http://participa.br/profiles/"+idd
        if metodo in ("topologico","hibrido"):
            # que seja de amigo ou de pessoa com quem interagiu
            if uri in d.nodes():
                x_n=d_[uri]
                x_n_=[(i,x_n[i]["weight"]) for i in x_n.keys()]
                x_n_.sort(key=lambda x: -x[1])
                # busca artigo destes participantes, recomenda os artigos
                count=0
                q=""
                for participante in x_n_[:5]:
                    q+="""SELECT ?artigo%i, abody%i
                        WHERE {
                            <%s> ops:performsParticipation ?artigo%i.
                             ?artigo%i schema:articleBody ?abody%i.
                            }
                        """%(count,count,participante[0],count,count,count
                    count+=1
                sparql=SPARQLWrapper(URL_ENDPOINT_)
                sparql.setQuery(PREFIX+q)
                sparql.setReturnFormat(JSON)
                results = sparql.query().convert()
                # adiciona recomendação
            if uri in g.nodes():
                vizinhos=g.degree(g.neighbors(uri))
                # adiciona recomendação
        if metodo in ("textual","hibrido"):
            # que tenha vocabulario parecido ou proximo
            pass

def recomendaComentario(destinatario, idd, metodo="hibrido", polaridade="m
    # que seja de amigo ou de pessoa com quem interagiu
    # que tenha vocabulario parecido ou proximo
    # que tenha maior media de ambas
    pass
def recomendaPalavra(destinatario, odd, metodo="hibrido", polaridade="mist
    pass
```

In [ ]: