Empoderando vidas.
Fortalecendo nações.

Projeto BRA/12/018 - Desenvolvimento de Metodologias de Articulação e Gestão de Políticas Públicas para Promoção da Democracia Participativa

Produto 04 - Proposta de adaptações e incrementos para a interface do portal federal de participação social e suas ferramentas

com elementos visuais e de usabilidade para mecanismos de priorização de conteúdos e autorregulação

Renato Fabbri

Secretaria-Geral da Presidência da República

Produto 04 - Proposta de adaptações e incrementos para a interface do portal federal de participação social e suas ferramentas

Contrato n. 2013/000566

Objeto da contratação: Aporte de conhecimentos e tecnologias para especificação de vocabulário e ferramentas assistidas que utilizam processamento de linguagem natural e análise de redes complexas para o conteúdo do portal da participação social.

Valor do produto: R$ 21,600 (vinte e um mil e seiscentos reais)

Data de entrega: 08 de Setembro de 2014

Nome do consultor: Renato Fabbri

Nome da supervisora: Gabriella Vieira Oliveira Gonçalves

Secretaria-Geral da Presidência da República

Fabbri, Renato

Proposta de adaptações e incrementos para a interface do portal federal de participação social e suas ferramentas: com elementos visuais e de usabilidade para mecanismos de priorização de conteúdos e autorregulação / 2014.

Total de folhas: 37

Supervisora: Gabriella Vieira Oliveira Gonçalves

Secretaria: SNAS

Secretaria-Geral da Presidência da República

Palavras-chave: reconhecimento de padrões, redes complexas, processamento de linguagem natural, participação social.

Sumário

Resumo

Este documento descreve rotinas de priorização de conteúdo e de autorregulação para o portal federal de participação social. Como parte da exposição e do Produto, foi implementado um sistema de recomendação de participantes para outros participantes e linha editorial do participa.br. Este sistema está online e visa atender a requisitos como os do ActionItem 3234 do Noosfero, que prevê recomendação de participantes, comunidades e empreendimentos. Como generalização da proposta, é delineada recomendação de recursos como trilhas, artigos, comentários e palavras. O consultor e equipe do participa.br entendem que a disponibilização destes recursos em formato aberto - e com documentações reativas para fácil geração de derivados, críticas e propostas - é instrumental para a democracia participativa e online, promovendo empoderamento e transparência.

Palavras-chave: reconhecimento de padrões, redes complexas, processamento de linguagem natural, participação social.

1 Introdução

1.1 Contexto e importância da consultoria

Em confluência com o portal federal de participação social (Participa.br) e o Plano Nacinal de Participação Social (PNPS), esta consultoria é um aporte de conhecimentos e tecnologias de web semântica, redes complexas e processamento de linguagem natural. O presente produto apresenta "formas de priorização de conteúdo e de autorregulação" na forma de um sistema de recomendação de recursos para participantes e linha editorial. No começo da seção 2 está explicitada a motivação para a implementação e a relação com o cumprimento dos termos deste Produto [1].

1.2 Contexto e importância do Produto

1.2.1 Objetivos

Este produto tem por objetivo principal a disponibilização de um sistema de recomendação de recursos do participa.br para usuários, tanto via critérios personalizados quanto considerando comunidades e linha editorial. Através deste sistema de recomendação, ficam facilitados, até mesmo prontamente disponíveis, diversos processos de autorregulação, de geração de resumos, de geração de relatórios e de análises informativas. Objetivos secundários são:

- Disponibilização de uma API HTTP, para uso no participa.br, conforme requisitado pela equipe do Participa.br em diversos itens dos produtos [2].
- A exposição destes algoritmos de recomendação aos visitantes web via uso de browsers comuns, para edição e execução dos trechos de código utilizados pela plataforma federal de participação social. Este objetivo foi satisfeito usando o IPython Notebook levantado para este trabalho [3].
- Aproveitamento do endpoint SparQL com os dados do participa.br, fortalecendo as tecnologias de dados linkados e web 3.0 [4].
- Em reunião com a consultora Daniela Feitosa, foi delineada a pertinência de um sistema de recomendação de perfis para um ActionIntem em andamento para o participa.br [5]. Este produto visa suprir esta necessidade através da API disponibilizada para recomendações.
- A entrega das tecnologias livres com simplicidade e boa documentação, favorecendo o aproveitamento deste trabalho para melhoras, geração de derivados e novas e independentes tecnologias. Isso pode ser observado no repositório git público deste produto [6].
- Compatibilizar ao máximo a entrega deste produto às demandas da equipe do participa.br [2].

- Realizar de forma precisa e pertinente a especificação deste quarto produto no Termo de Referência [1].

1.2.2 Resultados esperados

De imediato e mais central, o resultado do produto é iniciar um processo aberto de desenvolvimento e apropriação de análises e mecanismos de autorregulação para o portal federal de participação social.

Como resultados diretos deste Produto, constam:

- a habilitação para uso da API HTTP para recomendação de recursos do participa.br. Veja Apêndice A, D e E.
- A interface para apreensão e inovação dos algoritmos, não somente exemplificada ou projetada, mas operante e disponível. Veja Apêndice B.
- Um plano de implementação para o participa.br, que utiliza a API de recomendação para priorização de conteúdo e autorregulação. Veja Apêndice F.
- Transparência absoluta no trabalho, com toda a documentação e código computacional online em um repositório git que contem o histórico de implementação [6].
- Algoritmos implementados em código de simples leitura, para facilitar a implementação em outras linguagens como Ruby ou Javascript, quando houverem recursos mais maduros para estas linguagens ou por necessidade. No momento, Python possui mais e mais maduros recursos tanto para redes complexas quanto para processamento de linguagem natural.
- Habilitação de implementações em andamento, como o plugin de recomendação de perfis [5], para recomendar amigos para participantes.

1.2.3 Caráter inovador

Centralmente, este trabalho é inovador na aplicação de recursos de análise de redes sociais para empoderamento da sociedade civil, entregando as tecnologias e priorizando a reutilização. Este recurso é de vital interesse para a democracia participativa no contexto atual, com as revoluções da internet e das redes sociais. Permite, em última instância, que haja uma inteligência para aproveitamento das estruturas sociais, e que esta inteligência seja pública, transparente, minimizando vetores vigilantistas ou turvos.

Há a inovação na arquitetura em software, apresentando traços de web 3.0 como os dados linkados como base de conhecimento e os múltiplos recursos online acessados no funcionamento

usual (ao menos Endpoint SparQL para acesso aos dados, API HTTP Flask para tratar os dados e gerar estatísticas e estruturas de interesse, Participa.br para interface e contexto pertinente).

Há inovação na difusão científica. Além dos métodos e tecnologias disponibilizadas, por exemplo, os termos Web Semântica, Processamento de Linguagem Natural e Redes Complexas são empregados em textos científicos e consistem em áreas que recebem pesquisas, revistas e até carreiras científicas inteiras. Neste Produto, este conteúdo está apresentado em português e se presta a facilitar contribuições de outras partes interessadas.

Os métodos de recomendação em si não foram confrontados exaustivamente com a literatura, mas possivelmente possuem também inovações nos procedimentos. Certamente há inovação no contexto de implementação, tanto de relevância social (portal federal de participação social) quanto de aparato tecnológico (web 3.0, métodos da física e de mineração de dados).

1.2.4 Aparato em software e hardware

O sistema de recomendação precisa ser mantido online, e isso implica na manutenção de uma estrutura em hardware e software que extrapola o objeto deste produto. As especificações desta arquitetura de software e hardware estão em um documento escrito pelo consultor a pedido da PR para a SNAS e DITEC [7]. Para fins de pesquisa e de entrega deste produto, o esquema da Figura 1 é realizado em máquinas de pesquisa da USP, concedidas para a pesquisa de doutorado do consultor. Tal configuração é razoável pois a pesquisa conflui e é potencializada por esta consultoria, mas está prevista uma infraestrutura própria da PR para estes serviços. Além disso, os usos atuais ainda são moderados, sem causar sobrecarga ao serviços de computação em nuvem da USP.

2 Desenvolvimento

O produto é descrito no Termo de Referência desta consultoria assim: “Documento com proposta de adaptações e incrementos para a interface do portal e suas ferramentas que inclua elementos visuais e de usabilidade para mecanismos de priorização de conteúdos e auto-regulação com base nos metadados gerados nativamente pela plataforma do portal e nas análises de PLN e RC produzidas pelas ferramentas assistidas”.

Dada a dimensão do participa.br, tanto da estrutura em software e das práticas participativas, quanto de mobilização humana, as propostas de adaptações para o portal são muitas e estão em diversos produtos deste e de outros consultores. Uma forma especialmente pertinente de realizar este produto, confluente com o Termo de Referência desta consultoria, com o trabalho dos gestores e comunidade e com os produtos deste e de outros consultores [2], é a entrega de um sistema de recomendação de recursos para o participa.br.

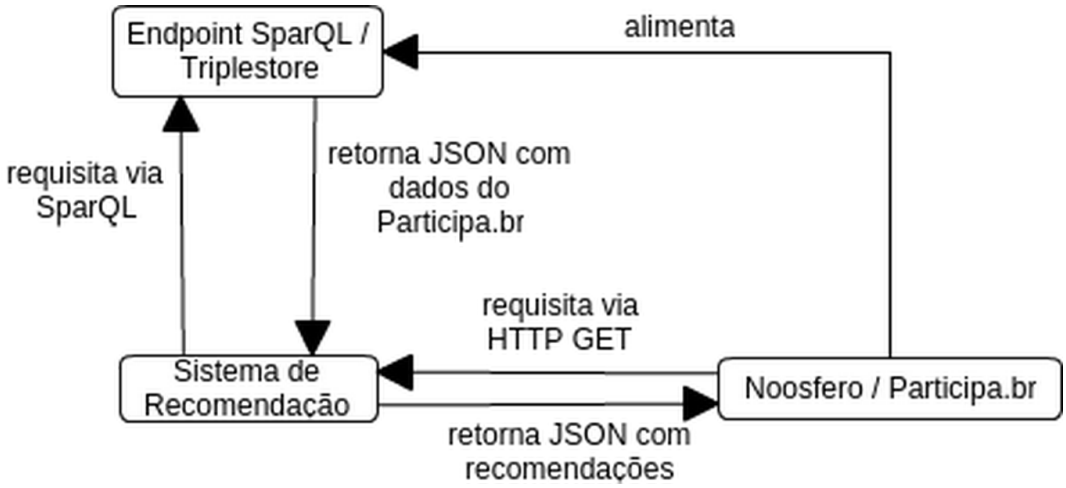

Figura 1: O sistema de recomendação e relações principais: com o frontend Participa.br e com a base de dados enriquecidos semanticamente.

Conforme discutido em reunião com gestores e comunidade do participa.br [8], em termos de *priorização de conteúdo*, quase todos os casos de interesse para o atual participa.br podem ser considerados rotinas de recomendação. Além disso, dadas as práticas atuais de redes sociais, os mecanismos de autorregulação do participa.br são e serão baseados na adição de conteúdo por espontaneidade dos participantes. Esta dinâmica de adição espontânea de conteúdo pelos participantes conflui com sistemas de recomendação de recursos, pois estes podem catalizar sem coerção os processos participativos.

Desta forma, foi idealizada a entrega de um sistema de recomendação de recursos do participa.br para seus usuários, comunidades e linha editorial. As subseções a seguir apontam etapas na construção do sistema de recomendação. Os Apêndices A e B concentram aspectos de operação do sistema. Os Apêndices D e E são voltados para os componentes e a instalação da versão operante deste sistema de recomendação de recursos. A seção 3 e o Apêndice F apreendem previsões e propostas de integração com o participa.br.

2.1 Etapas de desenvolvimento anteriores a este produto

2.1.1 Sistematização ontológica da participação online

Através de estudos e reuniões presenciais e online, a Ontologia de Participação Social (OPS) foi revisada [9] e a Ontologia do Participa.br (OPA) foi feita [10].

2.1.2 Triplificação dos dados do participa.br

Feito um script para triplificar os dados do Participa.br, ou seja, para o enriquecimento semântico e escrita em RDF dos dados em Postgresql da instância Noosfero do Participa.br [11].

2.1.3 Levantamento do endpoint SparQL

Para uso dos dados triplificados, pode-se recorrer a diversos métodos de leitura e disponibilização. Um método-chave é a disponibilização dos dados rdf (*triple store*) em um *endpoint sparql*. Para os fins de testes, pesquisa e usos leves, está disponibilizado um endpoint SparQL em servidores da USP [4].

2.1.4 Análises iniciais, modelos

Análises dos dados do participa.br foram abertas no IPython Notebook, com ênfase no texto produzido e nas redes formadas [12].

2.2 Etapas de desenvolvimento deste produto

2.2.1 Reuniões com equipe do participa.br

Especial agradecimentos ao Ricardo Poppi, Ronald Costa, Enaile Ladanza, Joênio Costa, Daniela Feitosa e Fernando Cruz.

2.2.2 Estudos de aprofundamento e amadurecimento

Em especial, foram lidos todos os produtos entregues pelos consultores, (re)visitados cursos no coursera e literatura científica [2].

2.2.3 Escrita deste documento

Através da escrita deste documento, várias informações foram organizadas e sistematizadas, facilitando compreensão do contexto para especificação da API de recomendação e entrega do produto.

2.2.4 Especificação da API de recomendação

Basicamente, é convencionado um padrão na montagem das URLs para recomendar recursos de tipos especificados, para destinatários e por métodos também especificados. O Apêndice A é dedicado a expor a convenção inicial entre URLs formadas e as recomendações que retornam.

2.2.5 Implementação do serviço para aquisição dos dados, processamento e entrega em JSON

Para a API da etapa acima, foi necessário desenvolver um servidor HTTP simples (em Flask). Veja pasta flask/ de [6].

2.2.6 Disponibilização das rotinas no IPython Notebook

Para expor os critérios de recomendação, disponibilizar análises e facilitar novas versões para as recomendações, as rotinas estão em um IPython Notebook. Veja o Apêndice B para uma exposição e detalhamento destas rotinas.

2.2.7 Proposta de implementações no portal federal de participação social

Com base no sistema de recomendações, nas demandas levantadas em reuniões e nos produtos já feitos por este e outros consultores, foi sistematizado um conjunto de propostas de acréscimos para o portal federal, detalhado no Apêndice F.

2.3 Justificativa do método

São usados diversos métodos neste produto. As justificativas se concentram nas tecnologias de web 3.0 (dados linkados e computação em nuvem), nas técnicas mais fundamentais/básicas/simples e nas demandas da equipe do participa.br, observadas através de reuniões e documentações produzidas recentemente pela equipe [2].

2.4 Justificativa das fontes

A equipe do participa.br é uma equipe estratégica da Presidência da República voltada para participação social. As fontes externas utilizadas, como artigos, livros e videos, são garimpadas como atual pesquisa acadêmica principal do consultor.

2.5 Confronto entre os resultados esperados e os alcançados

No Termo de Referência desta consultoria, este produto é descrito como "Documento com proposta de adaptações e incrementos para a interface do portal". Este produto entrega estas propostas a serem implementadas no portal federal de participação social. Além disso, para facilitar estas implementações e fazer a conexão com os desenvolvimentos ontológicos e de dados linkados, foi implementado e disponibilizado um sistema de recomendação de recursos para o participa.br. Um outro resultado para além do esperado, é a consideração cuidadosa da comunidade de democracia participativa, com a disponibilização das rotinas de recomendação no IPython Notebook [3] e uma visita aos produtos dos outros consultores [2].

3 Usos dos resultados

Há usos previstos do sistema de recomendação, pois são demandas da equipe que motivaram o seu desenvolvimento. Em especial, há o uso previsto no plugin de recomendação de perfis (ActionItem [5]).

Há usos próprios dos participantes, como consultas aos recursos recomendados para si e para outros, e usos para a linha editorial do portal. Estes usos podem ser feitos no próprio servidor que disponibiliza o serviço de recomendação. Podem ser adaptados para a interface do participa.br, com plugins ou temas apropriados. Algumas sugestões de implementações estão no Apêndice F.

Outro uso previsto, e que idealmente contará com incentivos da comunidade, é a evolução destes métodos de recomendação de perfis para melhor atender aos usos do portal e das comunidades. O consultor sugere que as comunidades sejam convidadas a apresentar uma ou mais pessoas com conhecimentos ou diposição para algoritmos e sistematizações. Esta pessoa poderia passar, por exemplo, por uma reunião para apreensão dos métodos de recomendação e análise, com vistas a proposição de melhoras e ampliação das funcionalidades.

As rotinas de recomendação estarão disponíveis online, e editáveis e executáveis no browser via IPython Notebook, o que deve facilitar dinâmicas de compartilhamento e amadurecumento destes processos. Este uso dos resultados deste produto é ainda mais central do que a API em si ou o uso dela dentro do participa.br. Mesmo assim, dada a complexidade das técnicas usadas, das tecnologias envolvidas e dos propósitos, o uso da API dentro do participa.br trará menos desafios que a apreensão, aproveitamento e melhora destes métodos pela comunidade de democracia participativa. Por isso, fica reforçada a pertinência da atenção da comunidade.

Um uso previso é a potencialização de uma biblioteca digital, incluindo buscas e recomendações no participa.br. Também sobre biblioteca digital, mas agora na alimentação dela, análises dos dados do participa.br podem ser feitas diariamente e constar como documentos da biblioteca digital. Outras análises especiais podem entrar como itens na biblioteca digital, facilitadas com as rotinas já disponibilizadas, e facilitando posteriores.

4 Conclusão

4.1 Comentários, sugestões, recomendações

O consultor recomenda que haja amadurecimentos em encontros, tanto para melhora deste legado do participa.br, quanto para apreensão dos métodos pelas comunidades interessadas.

As estruturas básicas para as análises e recomendações, que são as redes (de amizade e de interação) e os histogramas de palavras, podem ser convenientemente incluidos na triplificação dos dados, de forma que as recomendações e análises fiquem mais leves. Por hora, para viabilizar o uso, o sistema de recomendações refaz estas estruturas caso seja visitado o caminho `recomenda/atualiza`. Estas estruturas são:

- Rede de amizades.
- Rede de interação.
- Histograma de radicais das palavras de todos os textos (artigos e comentários) do participa.br.
- Seleção dos 400 radicais mais ocorrentes para caracterizar o domínio.
- Histograma de radicais de cada usuário e contagem das ocorrências das palavras selecionadas para caracterizar o domínio.

Os métodos de recomendação implementados utilizam separadamente critérios linguísticos ou de interação/relacionamento. Os casos e resultados já são interessantes e até complexos para uma primeira versão. Há implementações delineadas que utilizam ambos recursos linguísticos e de relacionamento (topológicos). Por hora, ambas as características podem ser utilizadas operando as pontuações das recomendações com base no texto com pontuações das recomendações com base em interação/relacionamento.

4.2 Impacto do Produto para a elaboração, gestão e/ou avaliação de políticas públicas de participação social

Este trabalho torna disponível uma porção de avaliações automáticas dos processos participativos que ocorreram ou ocorrerem no participa.br. Estas mesmas avaliações podem ser usadas continuamente, facilitando a gestão. A elaboração pode se beneficiar da análise das experiências passadas, facilitada pelos métodos aqui presentes.

Outro impacto pra a elaboração e gestão é basear o método de participação na utilização de métodos de recomendação de recursos. Por exemplo: pode-se recomendar que participantes com características X faça provocações. Estas provocações são enviadas para outros participantes

como recomendações. Ainda a outros participantes pode ser recomendada a sistematização destes resultados. Um exemplo mais próximo da democracia participativa atual é uma etapa em que os participantes escrevem para recomendados participantes para fazer contato e iniciar discussões.

Para a gestão, pode ficar facilitado o fomento e a qualificação da participação. Por exemplo, o plugin de recomendação de amigos [5] deixa o portal mais interessante para o participante, pois apreende melhor seus relacionamentos e expõe os critérios de busca como instrumentos para sua participação.

4.3 Como o Produto deverá impactar o público-alvo das políticas públicas a que se refere

As comunidades afeitas à participação social poderão aproveitar estas análises para melhor assimilação dos processos participativos, para relatórios, para gerar novos métodos de recomendação que melhor atendam aos interesses específicos das comunidades ou aos interesses da democracia participativa.

Um impacto imediato é a transparência reforçada dos processos participativos, pois não somente os dados, mas as análises e as recomendações dos recursos do participa.br estão online e publicamente disponíveis para geração de derivados.

Outro impacto imediato é a valorização da participação. Sempre que um participante, por exemplo, fizer uma amizade ou comentar uma postagem ou outro comentário, ele estará constando nas redes envolvidas, modificará as análises atuais, e poderá ver seu nome e de outros em relações diversas.

5 Agradecimentos

O consultor Renato Fabbri agradece ao Joenio Costa pelo template em LaTeXpara os produtos. Agradece à Daniela Feitosa pela reunião para demanda de recomendação de perfis. Agradece aos supervisores do trabalho realizado em torno do participa.br: Ricardo Poppi e Ronald Costa. Agradece ao labMacambira.sf.net e todas as comunidades de software e cultura livre que compõe esta contribuição.

Referências

[1] *Termo de Referência da consultoria a que se refere o presente produto.*

[2] *Repositório Git da leitura dos produtos dos outros consultores.* https://github.com/ttm/pnudExtra.

[3] *IPython Notebook com códigos dos produtos PNUD 3 e 4.* http://200.144.255.210:8080/.

[4] *Endpoint SparQL com dados do Participa.br.* http://200.144.255.210:8082/.

[5] *ActionItem do plugin de recomendação de perfis do participa.br.* http://noosfero.org/Development/ActionItem3234.

[6] *Repositório Git do produto 4: documento e scripts.* https://github.com/ttm/pnud4.

[7] "Especificação dos requisitos em hardware e software para o sistema de monitoramento do participa.br," http://tinyurl.com/m27wtfv.

[8] *Pad de proposta do produto 4: documento e scripts.* https://etherpad.mozilla.org/pnud4.

[9] "Ontologia de participação social," http://tinyurl.com/p2doueu.

[10] "Ontologia do participa," http://tinyurl.com/lcccwop.

[11] "Especificação da triplificação dos dados do participa.br," http://tinyurl.com/k74z3yl.

[12] *Repositório Git do produto 3: documento e scripts.* https://github.com/ttm/pnud3.

Abreviações e jargão

RC: Redes Complexas

PLN: Processamento de Linguagem Natural

OPS: Ontologia de participação Social

OPA: Ontologia do Participa.br

MMISSA: Monitoramento Massivo e Interativo da Sociedade pela Sociedade para Aproveitamento

AARS: A Análise de Redes Sociais

MyNSA: Monitoring yields Natural Streaming and Analysis

PNPS: Plano Nacional de Participação Social

RDF: Resource Description Framework

HTTP: Hypertext Transfer Protocol

SPARQL: Simple Protocol and RDF Query Language

endpoint SPARQL: ponto de acesso, geralmente HTTP, a dados em RDF via buscas em SPARQL.

Participa.br: Portal federal de participação social.

IPython Notebook: instância online para rodar scritps Python

Mateor: arcabouço para páginas reativas e com funcionamento distribuído.

D3js: biblioteca de visualização de dados.

A Especificação da API de recomendação de recursos do participa.br

As rotinas no Apêndice B possam ser adaptados para os mais diversos fins. Na API disponibilizada há quatro campos principais e dois auxiliares:

- Recurso: o recurso a ser recomendado: participantes, comunidades, trilhas, artigos ou comentários.
- Destinatário: para quem está sendo feita a recomendação: participante, comunidade ou linha editorial. Campo auxiliar "idd" para id do destinatário (comunidade ou participante).
- Método: método para a recomendação: "topologico", "textual" ou "hibrido". Campo auxiliar de polaridade similar, dissimilar ou mista.

A url usada para consulta HTTP incorpora os parâmetros da forma usual: `http://<urlDoServidor>/recomenda?recurso=participante&destinatario=linha_editorial&metodo=topologico&polaridade=mis&ordenacao=embaralhada`

B Rotinas de acesso e processamento de dados do participa.br para as recomendações

B.1 Estruturas auxiliares

Rotinas para obtenção das estruturas auxiliares para o sistema de recomendação

Estas rotinas são usadas para a rápida recomendação de recursos do Participa.br para usuários e para linha editorial.

Está prevista a disponibilização destas estruturas já no endpoint SparQL através de inclusão das rotinas na triplificação dos dados. Por hora, o cálculo é feito pelo sistema de recomendação. Estas são as estruturas atualizadas com a visita ao caminho http://"dominio"/atualiza

In [88]:
```
from SPARQLWrapper import SPARQLWrapper, JSON
import networkx as x, nltk as k
import string, time, __builtin__
import cPickle as pickle
```

In [5]:
```
URL_ENDPOINT="http://localhost:82/"
URL_ENDPOINT_=URL_ENDPOINT+"participabr/query"
EXCLUDE=set(string.punctuation+u'\u201c'+u'\u2018'+u'\u201d'+u'\u2022'+u'\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()
```

Rede de amizades

In [6]:
```
q="""SELECT ?a ?b ?aname ?bname
       WHERE {
          ?a foaf:knows ?b .
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
g=x.Graph()
for amizade in results["results"]["bindings"]:
    nome1=amizade["a"]["value"]
    nome2=amizade["b"]["value"]
    g.add_edge(nome1,nome2)
__builtin__.g=g
```

Rede de interação

In [9]:
```
q="""SELECT ?participante1 ?participante2 ?aname ?bname
       WHERE {
           ?comentario dc:type tsioc:Comment.
           ?participante1 ops:performsParticipation ?comentario.
           ?participante1 foaf:name ?aname.
           ?artigo sioc:has_reply ?comentario.
           ?participante2 ops:performsParticipation ?artigo.
           ?participante2 foaf:name ?bname.
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
d=x.DiGraph()
for interacao in results["results"]["bindings"]:
    nome_chegada=interacao["participante1"]["value"]
    nome_partida=interacao["participante2"]["value"]
    if (nome_partida,nome_chegada) in d.edges():
        d[nome_partida][nome_chegada]["weight"]+=1
    else:
        d.add_edge(nome_partida,nome_chegada,weight=1.)
__builtin__.d=d
```

Bag-Of-Words com todos os comentarios e artigos do participa.br

In [39]:
```
q="SELECT ?comentario ?titulo ?texto WHERE \
          {?comentario dc:type tsioc:Comment.\
         OPTIONAL {?comentario dc:title ?titulo . }\
          OPTIONAL {?comentario schema:text ?texto .}}"
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
msgs_=results["results"]["bindings"]
msgs=[mm for mm in msgs_ if ("titulo" not in mm.keys()) or
      (("teste de stress" not in mm["titulo"]["value"].lower())
      and ("comunidade de desenvolvedores e nesse caso, quanto mais"
           not in mm["texto"]["value"].lower()))]
palavras=string.join([i["texto"]["value"].lower() for i in msgs])
palavras = ''.join(ch for ch in palavras if ch not in EXCLUDE)
#palavras = ''.join(ch for ch in palavras if ch not in EXCLUDE).encode('ut
palavras_=palavras.split()
palavras__=[stemmer.stem(pp) for pp in palavras_ if pp not in STOPWORDS]
fdist_=k.FreqDist(palavras__)
# escolhendo as 400 palavras mais incidentes para referência
palavras_escolhidas=fdist_.keys()[:400]
__builtin__.palavras_escolhidas=palavras_escolhidas
__builtin__.fdist_=fdist_
```

In [100]:
```
T=time.time()
q="SELECT ?cbody ?titulo ?abody WHERE \
          {?foo ops:performsParticipation ?participacao.\
         OPTIONAL { ?participacao schema:articleBody ?abody. }\
         OPTIONAL {?participacao dc:title ?titulo . }\
```

```
            OPTIONAL {?participacao schema:text ?cbody .}}"
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
msgs_=results["results"]["bindings"]
msgs=[mm for mm in msgs_ if ("titulo" not in mm.keys()) or
      (("teste de stress" not in mm["titulo"]["value"].lower())
      or ("cbody" not in mm.keys() or ("comunidade de desenvolvedores e ne
           not in mm["cbody"]["value"].lower()))]
textos1=[i["cbody"]["value"] for i in msgs if "cbody" in i.keys()]
textos2=[i["abody"]["value"] for i in msgs if "abody" in i.keys()]
textos=textos1+textos2
# faz BoW e guarda num dict
texto=string.join(textos).lower()
texto_= ''.join(ch for ch in texto if ch not in EXCLUDE)

texto__=texto_.split()
#texto___=[stemmer.stem(pp) for pp in texto__]
texto___=[stemmer.stem(pp) for pp in texto__ if (pp not in STOPWORDS) and
bow=k.FreqDist(texto___)
radicais_escolhidos=bow.keys()[:400]
__builtin__.radicais_escolhidos=radicais_escolhidos
__builtin__.bow=bow
print("demorou %.2f segundos para fazer a BoW"%(time.time()-T,))
```

```
demorou 128.67 segundos para fazer a BoW
```

In [70]:
```
print(u"são %i textos postados e %i comentários,\n\
  totalizando %i palavras informativas em %i radicais"%
      (len(textos1), len(textos2),len(texto___), fdist.B()))
```

```
são 16253 textos postados e 1554 comentários,
  totalizando 2477698 palavras informativas em 18233 radicais
```

BoW de cada participante

In [105]:
```
T=time.time()
q="""SELECT DISTINCT ?participante
       WHERE {
          ?foo dc:contributor ?participante .
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
participantes_=results["results"]["bindings"]
participantes=[i["participante"]["value"] for i in participantes_]
# inicia loop
if "palavras_escolhidas" not in dir(__builtin__):
    print(u"rode BoW antes, para saber do vocabulário geral do portal")
else:
    palavras_escolhidas=__builtin__.palavras_escolhidas
bows={}
for participante in participantes:
    # puxa todos os comentarios de cada usuario
    # e os article bodys
```

```
    q="""SELECT DISTINCT ?abody ?cbody
         WHERE {
           <%s> ops:performsParticipation ?participacao.
             OPTIONAL { ?participacao schema:articleBody ?abody. }
             OPTIONAL { ?participacao schema:text ?cbody. }
             OPTIONAL {?comentario dc:title ?titulo . }
         }"""%(participante,)
    sparql = SPARQLWrapper("http://localhost:82/participabr/query")
    sparql.setQuery(PREFIX+q)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    results_=results["results"]["bindings"]
    results__=[mm for mm in results_ if ("titulo" not in mm.keys()) or
              (("teste de stress" not in mm["titulo"]["value"].lower()) or
              ("cbody" not in mm.keys() or
               ("comunidade de desenvolvedores e nesse caso, quanto mais"
                not in mm["cbody"]["value"].lower())))]

    textos1=[i["cbody"]["value"] for i in results__ if "cbody" in i.keys()
    textos2=[i["abody"]["value"] for i in results__ if "abody" in i.keys()
    textos=textos1+textos2
    # faz BoW e guarda num dict
    texto=string.join(textos).lower()
    texto_= ''.join(ch for ch in texto if ch not in EXCLUDE)
    texto__=texto_.split()
    texto___=[stemmer.stem(pp) for pp in texto__ if (pp not in STOPWORDS) a
    fdist=k.FreqDist(texto___)
    ocorrencias=[fdist[i] for i in radicais_escolhidos]
    bows[participante]=(fdist,ocorrencias)
__builtin__.bows=bows
len(texto___)
print("%.2f segundos para fazer BoW de cada participante"%(time.time()-T,)
```

```
895.30 segundos para fazer BoW de cada participante
```

In [103]:

```
pickle.dump( g, open( "pickledir/g.p", "wb" ) )
pickle.dump( d, open( "pickledir/d.p", "wb" ) )
pickle.dump( bow, open( "pickledir/bow.p", "wb" ) )
pickle.dump( radicais_escolhidos, open( "pickledir/radicais_escolhidos.p",
pickle.dump( bows, open( "pickledir/bows.p", "wb" ) )
```

B.2 Rotinas para recomendação de recursos

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 componen
                    # 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 [ ]:

C Parcela dos participantes que produziram texto ou interação

In [12]:
```
from SPARQLWrapper import SPARQLWrapper, JSON
import cPickle as pickle
```

In [3]:
```
URL_ENDPOINT="http://localhost:82/"
URL_ENDPOINT_=URL_ENDPOINT+"participabr/query"
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/>"""
```

In [19]:
```
q="""SELECT ?participant
       WHERE {
          <http://participa.br/> dc:contributor ?participant.
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
```

In [20]:
```
results_=results["results"]["bindings"]
participantes=[i["participant"]["value"] for i in results_]
```

In [21]:
```
len(participantes)
```

Out[21]: 3825

In [22]:
```
g=pickle.load( open( "pickledir/g.p", "rb" ) )
d=pickle.load( open( "pickledir/d.p", "rb" ) )
bows=pickle.load( open( "pickledir/bows.p", "rb" ) )
```

In [23]:
```
len(bows)
```

Out[23]: 3825

In [24]:
```
com_rastro=[part for part in participantes if
            (part in g.nodes()) or (part in d.nodes()) or (sum(bows[part][1])>0)]
```

In [25]:
```
len(com_rastro)
```

Out[25]: 872

In [32]:
```
print("%.2f por cento dos participantes são passíveis de serem recomendados e de rec
          ((len(com_rastro)/float(len(participantes)))*100,))
```

```
22.80 por cento dos participantes são passíveis de serem recomendados e de receberem
recomendacoes personalizadas
```

In [ ]:
```

```

D Infraestrutura do sistema de recomendações

A estrutura geral em software e hardware está delineada na subseção 1.2.4 e é tema de todo este documento. Este apêndice descreve os scripts do sistema de recomendação.

O sistema usufrui do endpoint SparQL para adquirir os dados. O usuário/cliente do sistema de recomendação acessa URLs via HTTP e recebe JSON, da forma usual. As rotinas de recomendação em si estão no Apêndice B. Dada a maturidade das bibliotecas em Python, a facilidade de desenvolvimento e a capacidade da comunidade em aproveitar scripts simples na linguagem, todo o sistema de recomendação está em Python. O servidor HTTP feito em Flask, o acesso ao endpoint SparQL (jena) pelo SPARQLWrapper, o processamento de linguagem natural em NLTK, os aproveitamentos das redes complexas em NetworkX. Bibliotecas padrão da linguagem (*builtin* na *Python Standard Library*), e que estão em uso no sistema de recomendação, são: json, cPickle, string, random, `__builtin__`.

O sistema está na pasta flask/ do repositório público [6]. As bibliotecas podem todas serem instaladas via pip e para iniciar o sistema, basta chamar `$ python recomenda.py` ou apontar o apache para a pasta via WSGI. O sistema levará até 30 minutos para iniciar, pois é necessária a criação das estruturas auxiliares descritas na seção 4.1. Este tempo de espera para processar é um dos principais motivadores para incluir estas estruturas nos dados triplificados, disponíveis no endpoint SparQL.

Pode-se considerar os arquivos do sistema de recomendação, disponíveis na pasta flask/, assim:

- `configuracao.py`: variáveis de configuração utilizadas em mais de um arquivo, como a URL do endpoint SparQL, as stopwords consideradas, os caracteres desconsiderados e os prefixos (namespaces) usados para as consultas SparQL. Atualmente o script possui menos de 20 linhas de código.

- `auxiliar.py`: rotinas de criação das estruturas auxiliares. Por hora são quatro: 1) redes de amizade e 2) de interação. 3) Histograma de palavras usadas em todo participa.br e 4) por cada usuário. Atualmente o script possui $\approx$ 150 linhas de código.

- `rotinasRecomendacao.py`: as rotinas de recomendação de recursos propriamente ditas. Utilizam as estruturas auxiliares e opções dadas pelo usuário/cliente. Retorna recomendações. Atualmente, o script possui mais de 350 linhas de código.

- `recomenda.py`: este arquivo levanta o servidor Flask e repassa as opções escritas na URL (método GET) para as rotinas em `rotinasRecomendacao.py`. Atualmente, o script possui menos de 100 linhas de código.

E Instalação e modificação do sistema de recomendações

O sistema de recomendação pode ser instalado em poucos passos. Por exemplo, em um sistema Ubuntu, os seguintes passos são suficientes:

1. Clone do repositório: `$ git clone http://github.com/ttm/pnud4`.
2. Na pasta flask/, inicia serviço com `$ python recomenda.py` (ou aponta apache para a pasta com wsgi).
3. Caso necessário, arrumar configurações de acesso ao endpoint SparQL no arquivo `configuração.py`.

Para fins de uso do sistema de recomendações dentro do Participa.br, é apropriado o uso de infraestrutura dedicada e que assegure a disponibilidade do serviço. Idealmente, este sistema deve estar junto and endpoint SparQL com os dados do participa.br. Para usos em frontends próprios, este mesmo sistema pode ser instalado em servidos de computação em nuvem gratuítos, como o heroku. Este uso de tecnologias leves e bastante difundidas visa facilitar a geração de materiais derivados. O IPython Notebook também se presta a este fim, com as rotinas expostas para execução e modificaçã pelo visitante que as acesso por browsers usuais (firefox, chrome) [3].

F Propostas de implementações na interface do portal federal de participação social

Apontar dev do meteor+d3 p dev, processamento e alocação distribuída, além de streaming.

Apontar ActionItem.

Visitar instancia online do participa.br para apontar na interface possibilidades de implementação de outras recomendações e análises.

Apontar da criação de análises periódicas diárias, semanais, mensais, etc.

Apontar o uso do sistema de recomendações para recursos da biblioteca digital e para integrar o portal como um todo.