Anexo A: Classificação de conteúdo via mensagens rotuladas e personas

Importação das bibliotecas para acesso aos dados enriquecidos semanticamente:

In [70]:
```
from SPARQLWrapper import SPARQLWrapper, JSON
import time
```

Definição de prefixos úteis para as consultas SPARQL:

In [71]:
```
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 schema: <http://schema.org/>"""
```

Buscando todos os comentários no endpoint SparQL Fuseki/Jena:

In [72]:
```
NOW=time.time()
q="SELECT ?comentario ?titulo ?texto WHERE \
           {?comentario dc:type tsioc:Comment.\
          OPTIONAL {?comentario dc:title ?titulo . }\
           OPTIONAL {?comentario schema:text ?texto .}}"
sparql3 = SPARQLWrapper("http://localhost:82/participabr/query")
sparql3.setQuery(PREFIX+q)
sparql3.setReturnFormat(JSON)
results4 = sparql3.query().convert()
print("%.2f segundos para puxar todos os comentários do Participa.br"%
        (time.time()-NOW,))
```

```
2.43 segundos para puxar todos os comentários do Participa.br
```

Removendo pontuação e fazendo lista de palavras. São muitas mensagens repetidas, dois tipos:

- uma possui "teste de stress" no título, outra possui
- "comunidade de desenvolvedores e nesse caso, quanto mais" no corpo da mensagem.

Ambas as mensagens são removidas.

In [87]:
```
msgs_=results4["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()))]
NOW=time.time()
import string, nltk as k
exclude = set(string.punctuation+u'\u201c'+u'\u2018'+u'\u201d'+u'\u2022'+u'\u2013')
palavras=string.join([i["texto"]["value"].lower() for i in msgs])
palavras = ''.join(ch for ch in palavras if ch not in exclude)
palavras_=palavras.split()
print(u"feita lista de todas as palavras de todos os comentários em %.2fs"%
      (time.time()-NOW,))
```

```
feita lista de todas as palavras de todos os comentários em 0.19s
```

Removendo stopwords e fazendo contagem das palavras restantes:

In [83]:
```
NOW=time.time()
stopwords = set(k.corpus.stopwords.words('portuguese'))
palavras__=[pp for pp in palavras_ if pp not in stopwords]
fdist_=k.FreqDist(palavras__)
print("retiradas stopwords e feita contagem das palavras em %.2fs"%
      (time.time()-NOW,))
for fd,ii in [(fdist_[i],i) for i in fdist_.keys()[:14]]: print fd, ii
```

```
retiradas stopwords e feita contagem das palavras em 0.29s
1277 é
1256 não
762 ser
717 participação
548 social
526 sociedade
468 à
459 sobre
367 governo
357 são
337 forma
327 políticas
310 públicas
302 brasil
```

In [75]:
```
print(u"são %i palavras em %i palavras diferentes"%(len(palavras__),len(fdist_)))
```

```
são 91361 palavras em 14653 palavras diferentes
```

In [76]:
```
# para radicalizar (lematização é similar)
# NOW=time.time()
#stemmer = k.stem.RSLPStemmer()
#palavras___=[stemmer.stem(pp) for pp in palavras__]
#fdist__=k.FreqDist(palavras___)
#print("feita freq dist (radicalizada) em %.2f"%(time.time()-NOW,))
```

Escolhendo as palavras mais frequentes para fazer caracterização das mensagens:

In [77]:
```
# escolhendo as 200 palavras mais frequentes
palavras_escolhidas=fdist_.keys()[:200]
```

Extraindo atributos (contagem das palavras) e fazendo classificação bayesiana ingênua. Note que os rótulos "pos" e "neg" estão sendo atribuídos ao acaso. Para aproveitamento, é necessário que sejam usados os dados rotulados, provavelmente rotulados pelo pessoal da comunicação.

In [78]:
```
def document_features(documento):
    features={}
    for palavra in palavras_escolhidas:
        features["contains(%s)"%(palavra,)]=(palavra in documento)
    return features
msgsP= [(rr["texto"]["value"],"pos") for rr in msgs[:500]]
msgsN=[(rr["texto"]["value"],"neg") for rr in msgs[500:1000]]
msgsT=msgsP+msgsN
random.shuffle(msgsT)
feature_sets=[(document_features(msg[0]),msg[1]) for msg in msgsT]
train_set, test_set = feature_sets[:500], feature_sets[500:]
```

```
classifier = k.NaiveBayesClassifier.train(train_set)
```

Mostrando as características mais informativas:

In [79]:
```
classifier.show_most_informative_features(5)
```

```
Most Informative Features
        contains(comitê) = True              pos : neg    =      4.1 : 1.0
          contains(hoje) = True              pos : neg    =      4.0 : 1.0
         contains(saúde) = True              neg : pos    =      3.1 : 1.0
        contains(sugiro) = True              neg : pos    =      2.7 : 1.0
         contains(grupo) = True              pos : neg    =      2.4 : 1.0
```

Precisão nos dados de teste dummy ≈ 0.5, pois os rótulos não foram atribuídos com coerência (classificações de antemão de um conjunto de mensagens por conhecedores):

In [80]:
```
k.classify.accuracy(classifier, test_set)
```

Out[80]:
```
0.526
```

Classificação de um documento:

In [86]:
```
classifier.classify(document_features(msgsT[12][0]))
```

Out[86]:
```
'neg'
```

|||--- FIM ---|||