

Curso Ciência de Dados e Big Data – Turma 2
Campus Praça da Liberdade



Pontifícia Universidade Católica de Minas Gerais

Trabalho final da disciplina de Recuperação da Informação na Web e em
redes sociais:

**Análise de sentimento e geolocalização de *replies*
postados no Twitter**

Rodrigo Carlos de Jesus Teodoro

Professor: Cristiano Carvalho

Belo Horizonte
2017

Índice

Introdução	3
Recuperação dos dados	4
Configuração das API's do Python para integrar com o Google Maps	6
Configuração do workflow do Knime	6
Dados gerados	11
Análise	15
Conclusão	15
Anexos	17
Anexo 1 - Recuperar replies	17
Anexo 2 - pré-processamento dos tweets	18
Anexo 3 - extração das palavras positivas e negativas	19
Anexo 4 - Resultado API geocode	19
Anexo 5 - Separar texto e localização	20
Anexo 6 - Separar palavras por localidade	21
Anexo 7 - Utilização da biblioteca Flask Google Maps - usuários únicos e distribuídos por partido	23
Anexo 8 - Utilização da biblioteca Flask Google Maps - localização das palavras	24

Introdução

Este trabalho tem como objetivo demonstrar os conhecimentos aprendidos na disciplina de recuperação da informação da WEB e redes sociais num projeto que recupera e analisa os *replies* de *tweets*, coletados entre março a agosto de 2017, que foram enviados em resposta aos *tweets* publicados pelos três principais partidos políticos em foco no Brasil neste ano de 2017, sendo:

- PSDB - <https://twitter.com/rede45>
- PMDB - https://twitter.com/PMDB_Nacional
- ptbrasil - <https://twitter.com/ptbrasil>

Sobre a base de *replies* recuperados serão feitas as seguintes análises:

- Sentimento (positivo ou negativo) - o que os usuários do Twitter colocaram nos replies, gerando uma nuvem de palavras colorida de acordo com o sentimento provocado.
- Geolocalização das palavras no mapa do Brasil, sendo:
 - Geolocalização das pessoas que enviaram os *replies* no mapa do Brasil
 - Geolocalização de cada palavra contida nos *replies* com seu sentimento no mapa do Brasil

O motivo de recuperar os *replies* de *tweets* é que podemos considerar que o que foi postado pelo partido fez com o usuário se sentisse mais motivado em escrever uma resposta mais complexa e que possa expressar o seu sentimento sobre o assunto.

O classificador que será utilizado para definir os sentimentos das palavras será o dicionário Brazilian Portuguese LIWC 2007 (<http://143.107.183.175:21380/portlex/index.php/pt/projetos/liwc>), ele contém mais de 24 mil palavras classificadas como positivas e negativas.

Para realizar o trabalho, foram utilizados :

- Linguagem de programação Python 3.5 - para recuperação do replies, tratamento pré-processamento dos textos e integração a as apis de geolocalização e mapas do Google.
- Para o fazer o pré-processamento da geolocalização dos *replies*, será utilizada a *Google Maps Geocoding API* que é um serviço que oferece geocodificação e geocodificação reversa de endereços.
- Para validar se a palavra está escrita corretamente dentro do texto será utilizada a biblioteca *PyEnchant* (<http://pythonhosted.org/pyenchant>) do Python. Essa biblioteca funciona em Linux, no caso, foi utilizado Ubuntu 16.04
- *Knime* 2.12.2 para Windows - software Open Source para análise de dados, que irá gerar a nuvem de palavras com os devidos sentimentos.

Os arquivos fontes em Python quando o *workflow* do *Knime* são disponíveis no perfil do autor desse trabalho no GitHub: <https://github.com/rodrigoteodoro/ritrabalhofinal>

Recuperação dos dados

- Recuperação dos *tweets*

Utilizando a api do Twitter, foram resgatados o *tweets* por período e os textos já passam por um tratamento inicial removendo emojis, hashtags e caracteres estranhos.

Importante ressaltar que a api do Twitter tem limitação de 180 chamadas em 15 minutos, então é importante ficar atento a esses limites quando se tenta recuperar um número grande de tweets, veja mais em: https://dev.twitter.com/rest/reference/get/application/rate_limit_status.

Fonte: [Anexo 1 - Recuperar replies](#)

Exemplo do arquivo em formato csv gerado:

```
"partido";"userid";"username";"location";"texto";"coordinates"
"Rede45";"2801814928";"Elcio Teixeira";"Cortês, PE Brasil";" maior melhor renovação política seria extinção desse partido tenham vergonha ajudam temer acabar";""
```

<https://github.com/rodrigoteodoro/ritrabalhofinal/blob/master/rirabalhofinal/dados/amostrainicialtexto.csv>

- Pré-processamento dos *tweets*

Após a coleta e armazenamento dos tweets, será realizada uma nova leitura dos textos removendo palavras que foram escritas de forma incorreta pelos usuários e palavras que não existem no dicionário.

Essa etapa é fundamental para que o processamento de sentimentos e classificação seja mais rápida, já que não irá testar palavras desnecessárias.

Fonte: [Anexo 2 - Pré-processamento dos tweets](#)

A biblioteca *PyEnchant* pode buscar sugestões de uma palavra escrita de forma incorreta, porém é necessário criar um algoritmo próprio para analisar cada palavra sugerida para tentar substituir a palavra escrita de forma incorreta na frase, mas não utilizei esse recurso pois seria muito complexo no momento.

- Extração das palavras positivas e negativas do Brazilian Portuguese LIWC 2007 Dictionary

O dicionário Brazilian Portuguese LIWC 2007, possui a classificação das palavras em português como: positivas, negativas, pronome, verbos, biologia, saúde, entre outros, como por exemplo:

	humans	affect	posemo	percept	see	achieve
alegria	124	125	126	140	141	355

Veja que a palavra acima, possui em sua linha os números relacionados a sua classificação separados por tabulação. Para que esse arquivo seja utilizado no *Knime* e na classificação para a

exibição no mapa, foi necessário criar uma função no *Python* que faça a extração das palavras que tem classificação 126 = posemo (positiva) e 127 = negemo (negativa).

Fonte: [Anexo 3 - extração das palavras positivas e negativas](#)

Exemplo do arquivo em formato CSV gerado:

```
"palavra";"sentimento"
"abafa";"neg"
```

https://github.com/rodrigoteodoro/ritrabalhofinal/blob/master/ritrabalhofinal/dados/corpus_positivo_negativo_from_LIWC2007_.csv

- Pré-processamento da localização dos *tweets*

De posse da base de *tweets* serão separados os registros que contém a localização do usuário que fez o *post* e através da *Google Maps Geocoding API*, será passado o que foi descrito como localização pelo usuário para que se tenha o nome correto do local e a latitude e longitude.

Essa api é interessante pois ela reconhece algumas formas de descrever o lugar, como por exemplo: sampa, e retornar corretamente como São Paulo.

Veja que no campo **formatted_address** temo o valor "**São Paulo, SP, Brasil**", e em **location** os valores de latitude e longitude.

<https://maps.googleapis.com/maps/api/geocode/json?address=sampa>

Fonte: [Anexo 4 - Resultado API geocode](#)

Abaixo temos o fonte que irá, para cada *tweet*, buscar a localização e atualizar o registro os locais que estão no Brasil.

Fonte: [Anexo 5 - Separar texto e localização](#)

Exemplo do arquivo em formato CSV gerado:

```
"location";"formatted_address";"lat";"lng";"lat_lng"
"Cortês, PE Brasil";"Cortês - State of Pernambuco, Brazil";-8.4257032;-35.5329497;"-8.4257032,-35.5329497"
```

<https://github.com/rodrigoteodoro/ritrabalhofinal/blob/master/ritrabalhofinal/dados/localidades.csv>

Depois será feita a classificação do sentimento e separação das palavras e localização pelo fonte abaixo:

Fonte: [Anexo 6 - Separar palavras por localidade](#)

Exemplo do arquivo em formato CSV gerado:

```
"partido";"palavra";"tipo";"lat";"lng"
"PMDB_Nacional";"corrupto";"neg";"-12.9722184";"-38.5014136"
"PMDB_Nacional";"aprovada";"pos";"-21.2633142";"-48.3103085"
```

https://github.com/rodrigoteodoro/ritrabalhofinal/blob/master/ritrabalhofinal/dados/palavras_location_separadas2.csv

Configuração das API's do Python para integrar com o Google Maps

Para geolocalizar as palavras e os usuários no Google Maps, foi necessário utilizar as seguintes bibliotecas do Python:

- Flask - micro framework para desenvolvimento web em Python, serve para criar webserver, possuindo a implementação básica para interceptar requests e lidar com response, controle de cache, cookies, status HTTP, roteamento de urls e também conta com uma poderosa ferramenta de debug.
<http://flask.pocoo.org/>
- Flask Google Maps - utilizada para fazer a integração com o Google Maps em uma página da web.
<https://github.com/rochacbruno/Flask-GoogleMaps>

Então abaixo temos a configuração do Flask e Flask Google Maps passando a chave (parâmetro *key*) da API do Google. Essa chave é necessária quando se utiliza mais um mapa na mesma página ou se tem muitas requisições no seu projeto.




```
app = Flask(__name__, template_folder=".")
GoogleMaps(app, key="")
```

Para esse trabalho, foi criado dois tipos de visualização:

- Localização dos usuários únicos e distribuídos por partido, sendo o fonte responsável abaixo pela tarefa:

Fonte: [Anexo 7 - Utilização da biblioteca Flask Google Maps - usuários únicos e distribuídos por partido](#)

Pode-se notar que cada partido terá uma marcação de cor diferenciada, sendo:

PSDB 	PT 	PMDB 
---	---	---

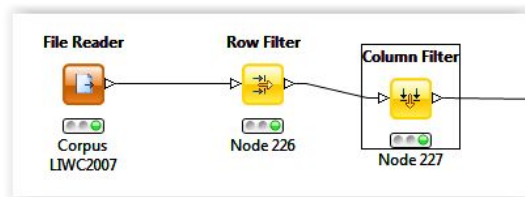
- Localização das palavras distribuídas em três mapas por partido e ao lado a quantidade palavras positivas e negativas, sendo o fonte responsável abaixo pela tarefa:

Fonte: [Anexo 8 - Utilização da biblioteca Flask Google Maps - localização das palavras](#)

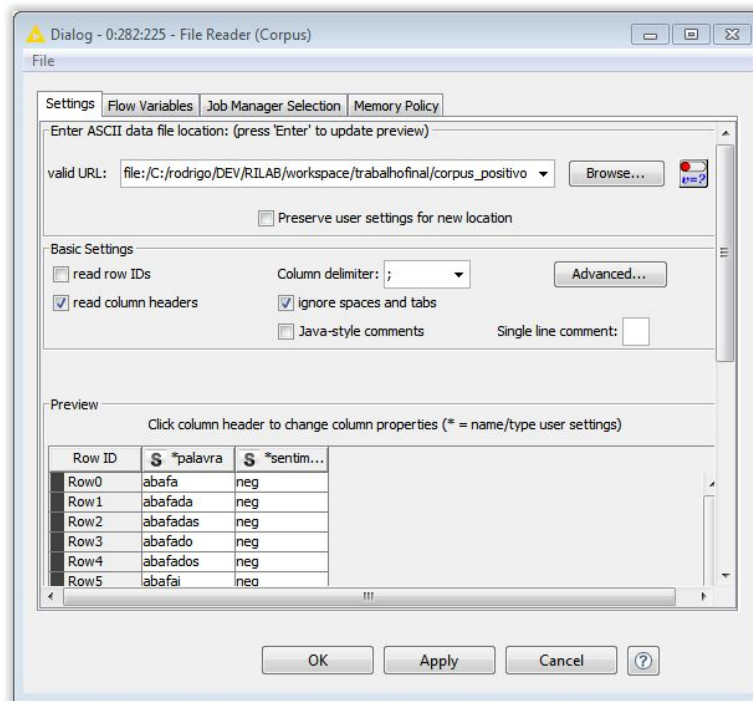
Configuração do *workflow* do *Knime*

Para que a ferramenta possa gerar a nuvem de palavras e classificar por cores os sentimentos delas, foi necessário configurar fluxo como abaixo:

- Palavras positiva retiradas do Brazilian Portuguese LIWC 2007 Dictionary

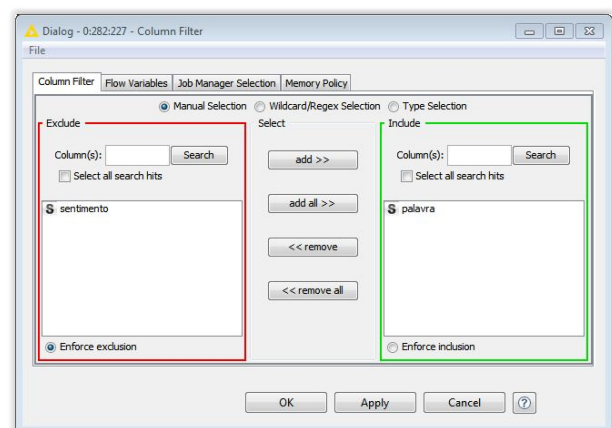
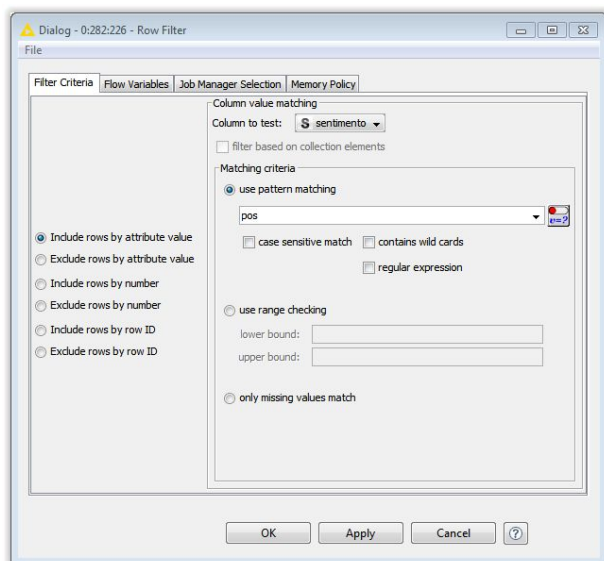


File Reader: Ler o arquivos com o corpus extraído do LIWC 2007 Dictionary

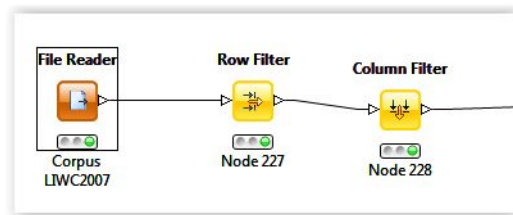


Row Filter: Filtrar os registros das palavras que são positivas

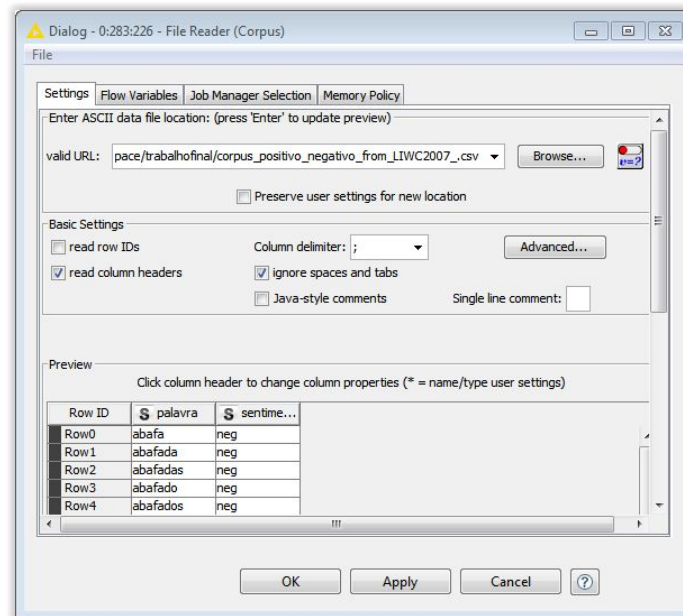
Column Filter: Separar a coluna das palavras que são positivas



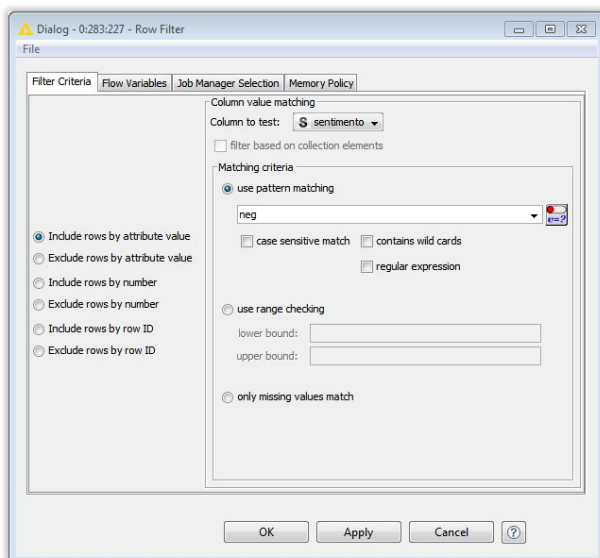
- Palavras negativas retiradas do Brazilian Portuguese LIWC 2007 Dictionary



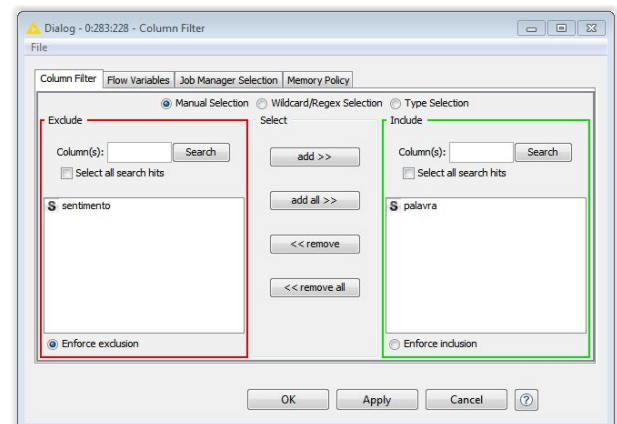
File Reader: Ler o arquivos com o corpus extraído do LIWC 2007 Dictionary



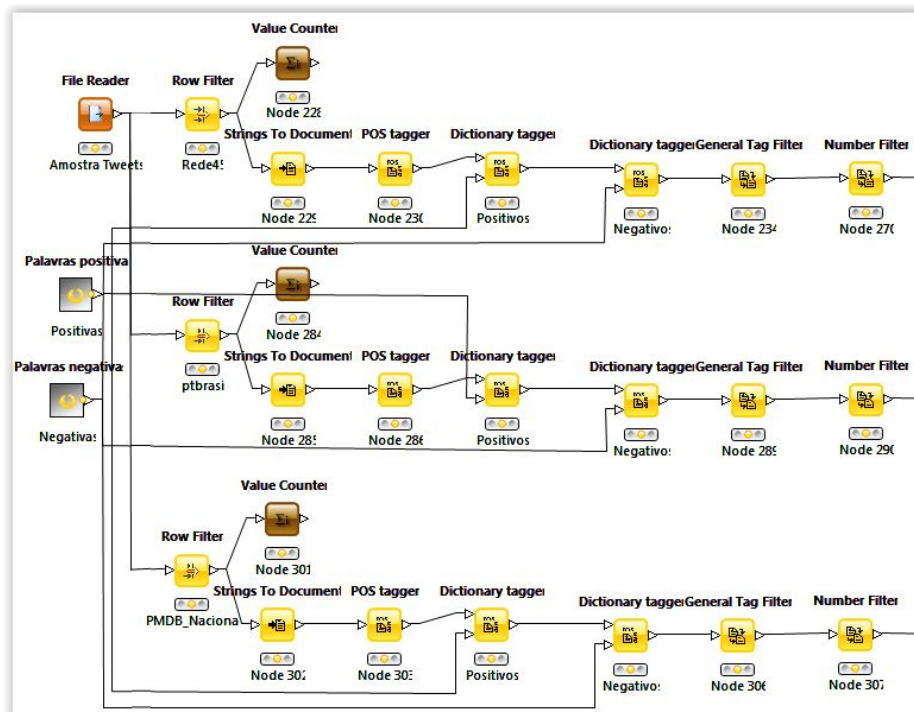
Row Filter: Filtrar os registros das palavras que são negativas



Column Filter: Separar a coluna das palavras que são positivas

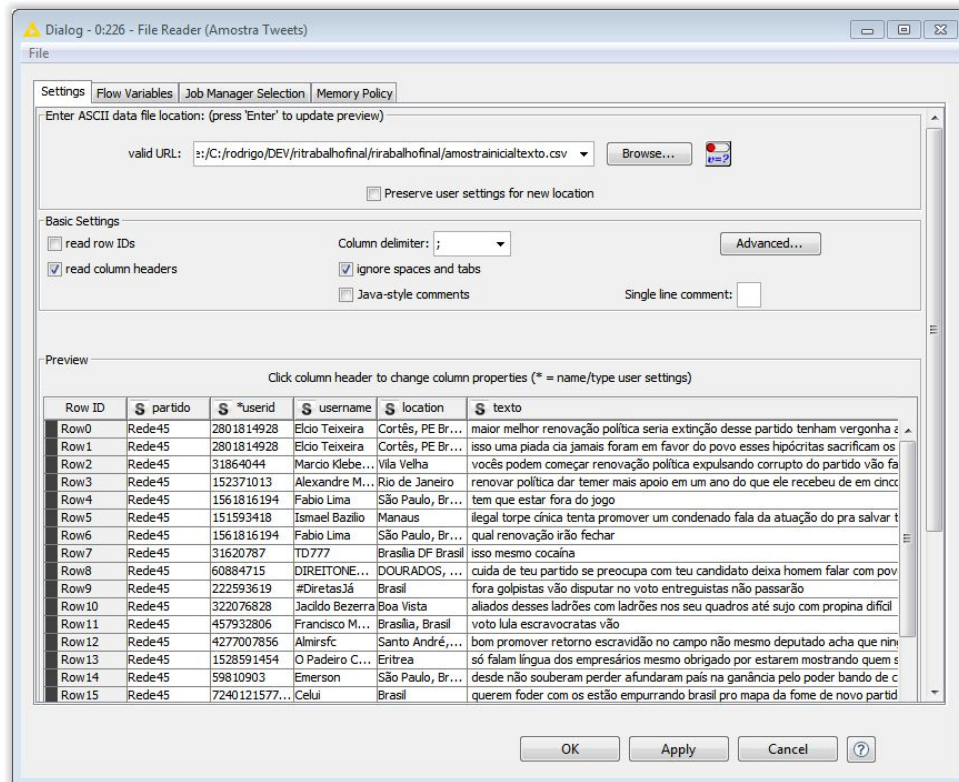


- Abaixo a visão do fluxo completo:

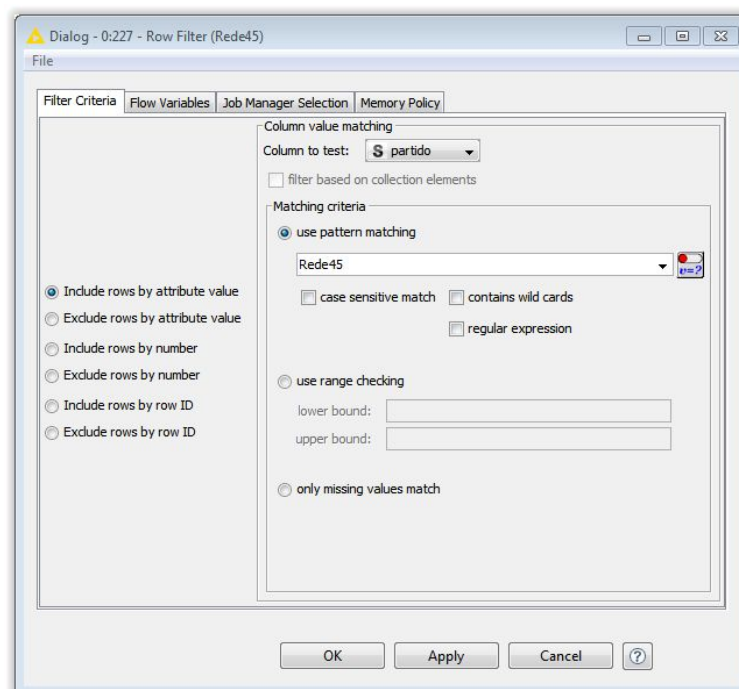


- Separação do tweets por partido:

File reader - Leitura do arquivo com os tweets



Row Filter - Separar os registros de cara partido:

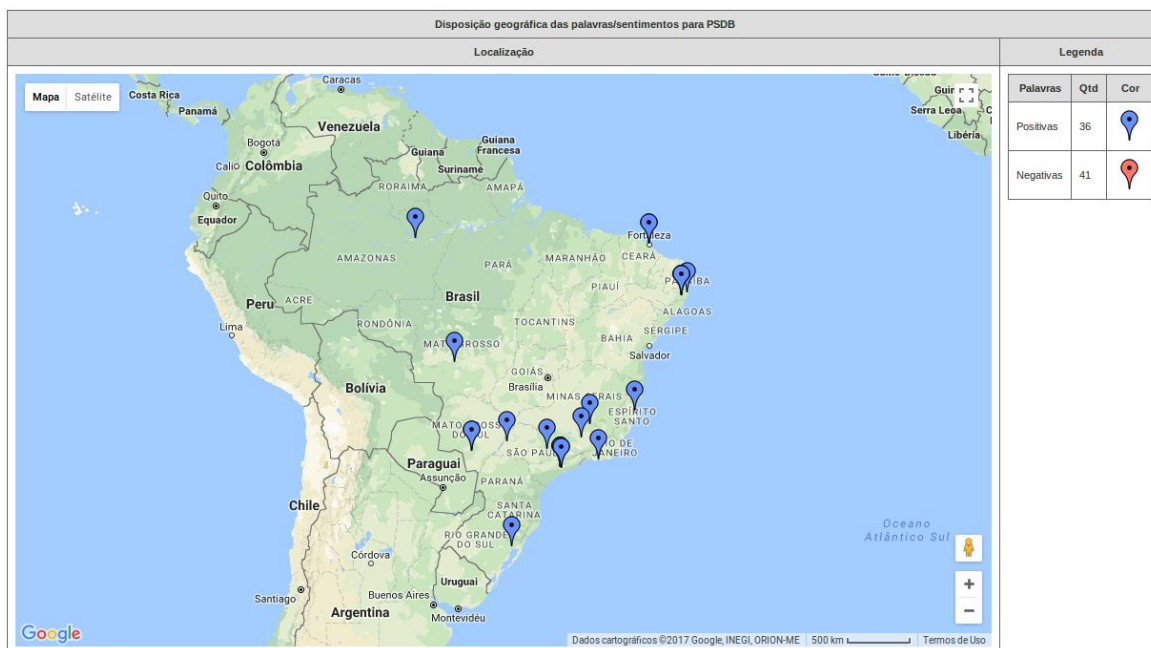
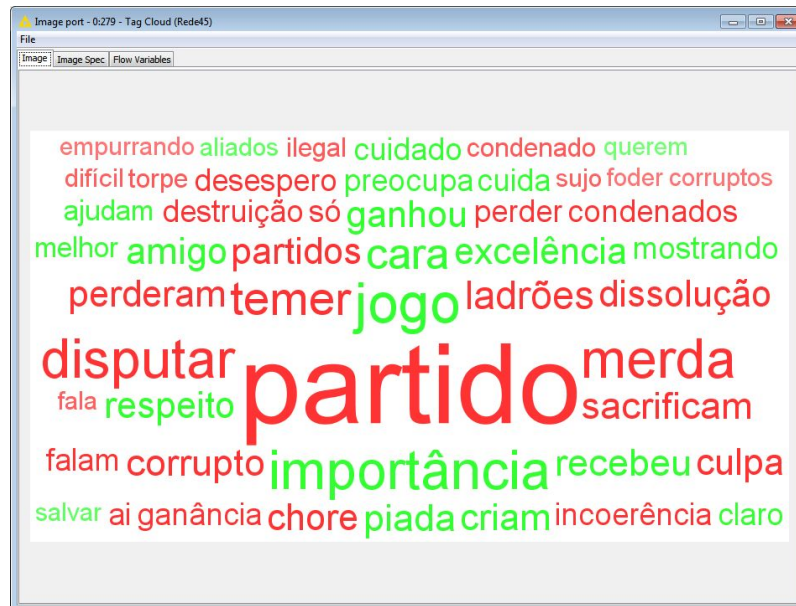


Os demais componentes do fluxo tem configurações simples, maioria padrão do componente e que podem ser vistas mais detalhadamente baixando o workflow em: <https://github.com/rodrigoteodoro/ritrabalhofinal/blob/master/knime/trabalhofinal.zip>

Dados gerados

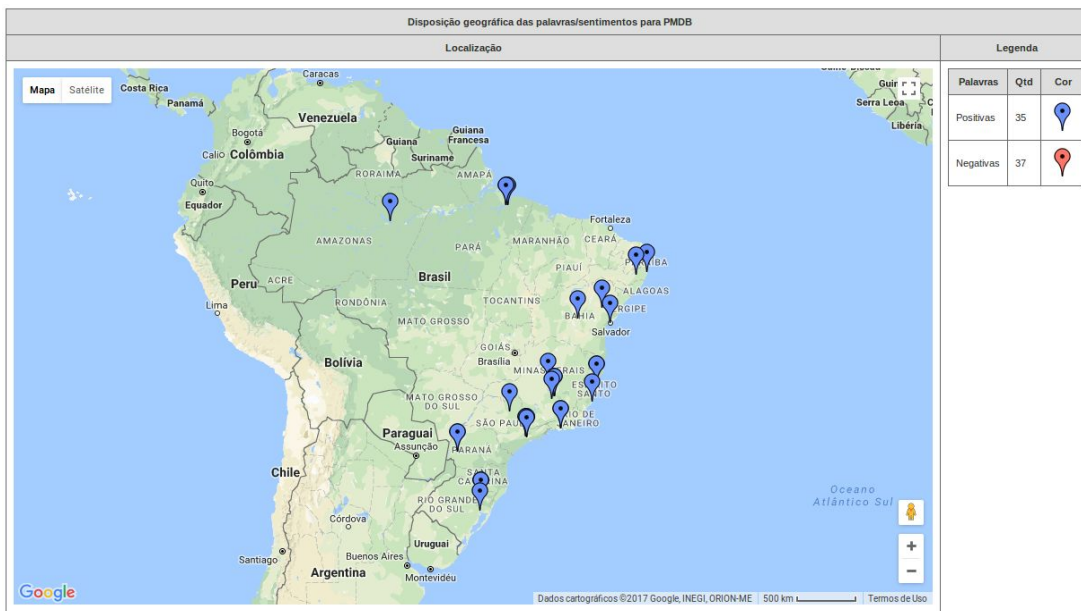
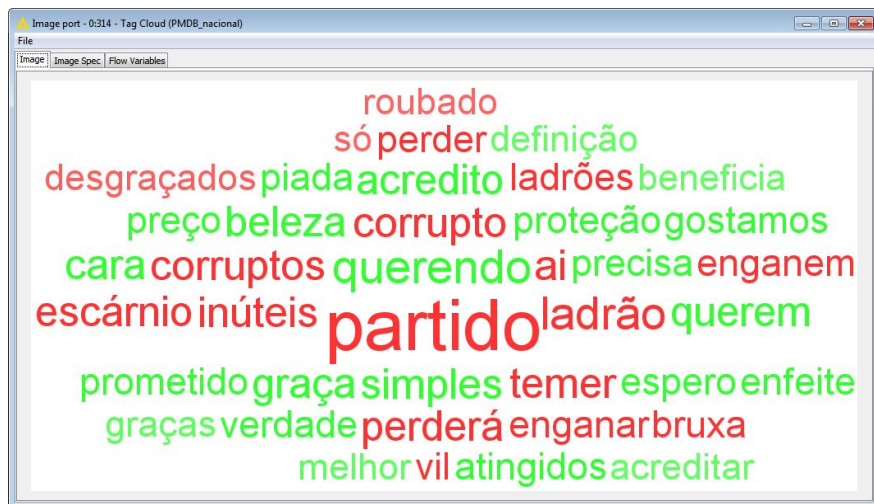
Abaixo temos as nuvens de palavras geradas por partido, sendo que em verde são palavras positivas e vermelho as negativas e a geolocalização das palavras no Google Maps.

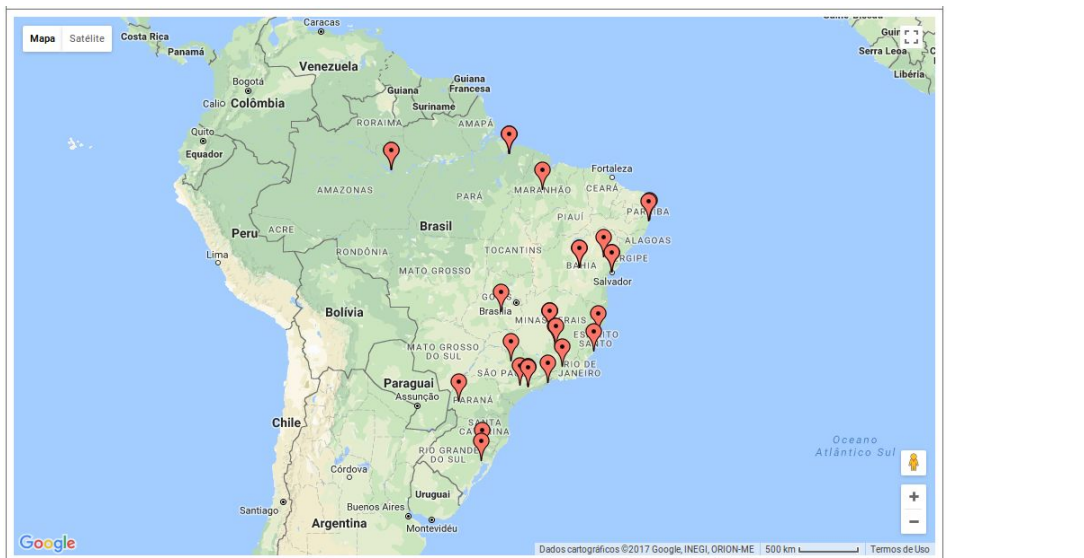
- PSDB



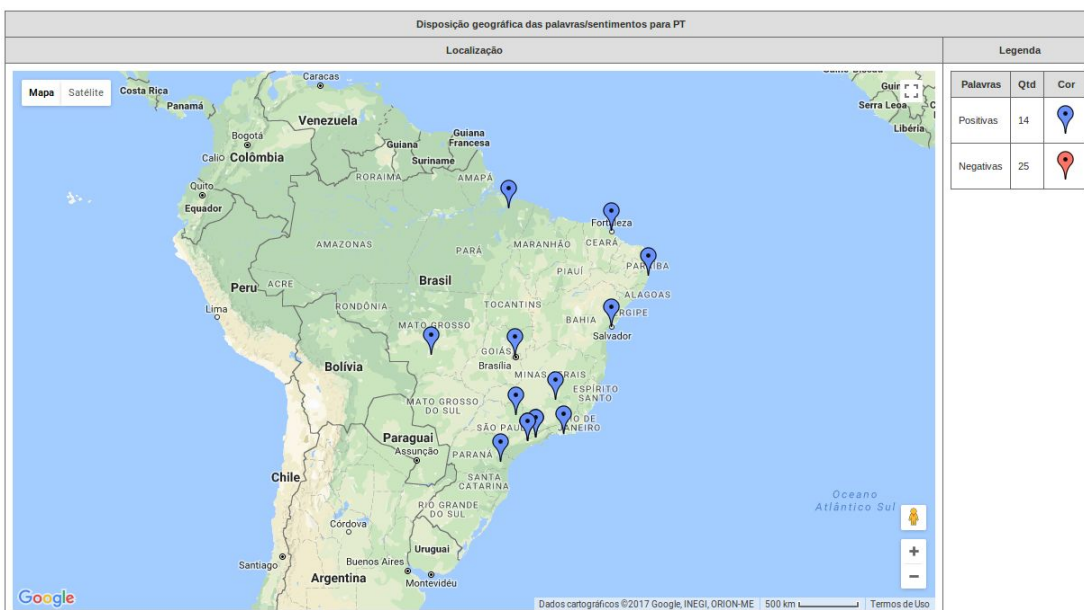
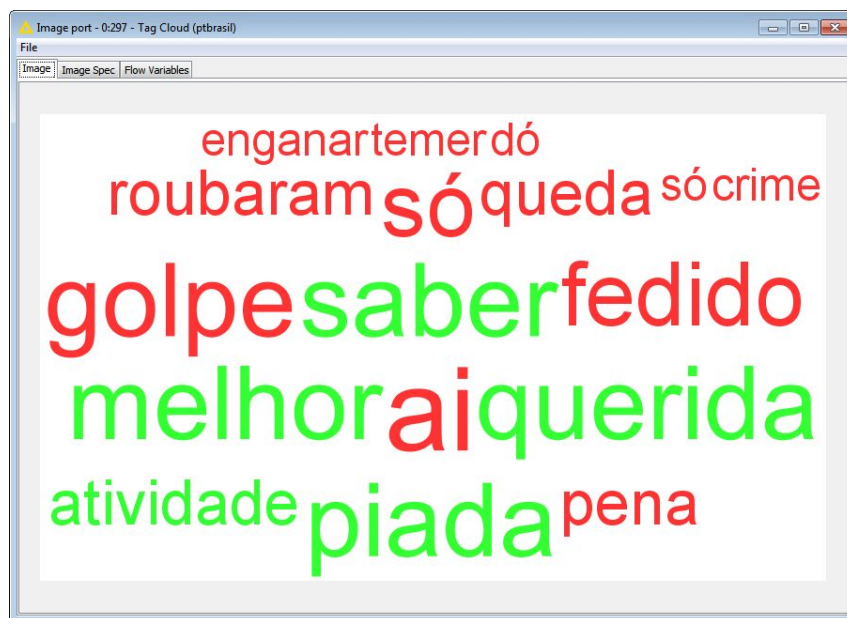


- PMDB





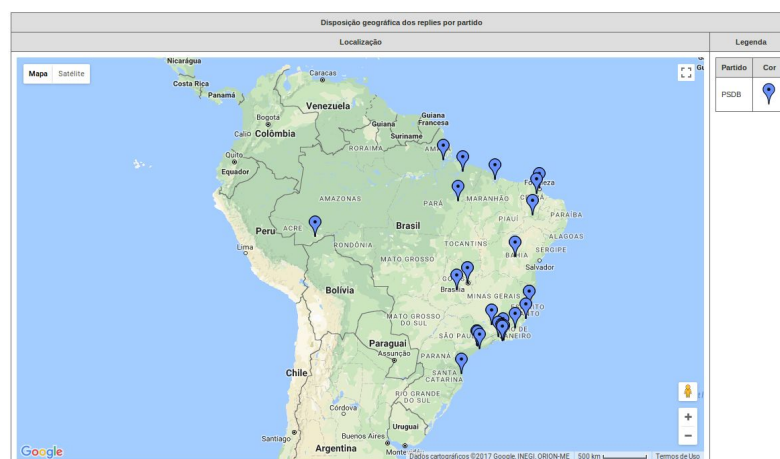
- PT



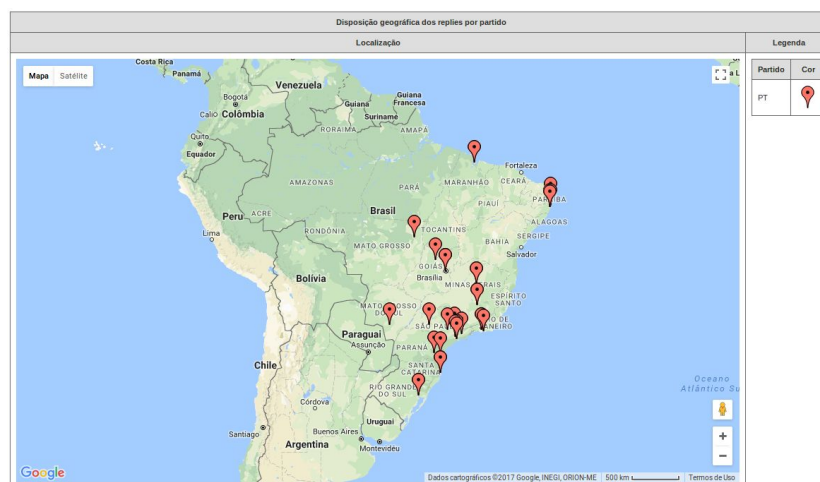


- Abaixo temos a geolocalização da influência de onde cada partido gerou uma resposta do usuário:

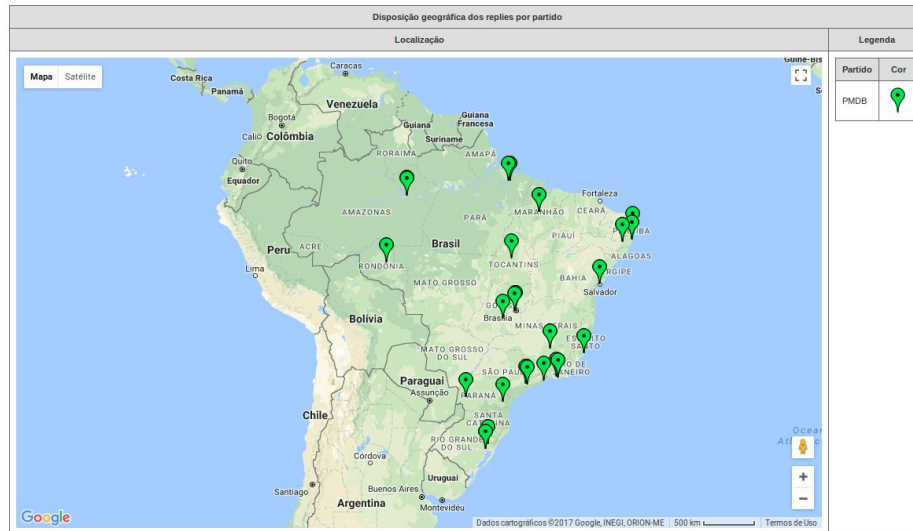
- PSDB



- PT



- PMDB



Análise

No período da amostra de março a agosto de 2017 pode-se notar a influência de cada partido com se segue:

- PSDB - maior quantidade de *replies* na região sudeste e devido ao momento atual um pouco mais de palavras negativas com maior concentração no nordeste.
A palavra que mais repete é partido, porém se referente ao partido político e não ao ato de partir com contexto negativo, logo podemos destacar as palavras mais negativas como: merda, sacrificam e dissolução. Respeito e excelência são as palavras mais positivas.
- PT - maior quantidade de *replies* na região de São Paulo e Nordeste e devido ao momento atual uma diferença significativa e maior de palavras negativas no sudeste.
As palavras que mais se repetem são negativas, fedido e golpe. As palavras piada e querida, apesar de estarem marcadas como positivas, nesse contexto atual tem conotação negativa.
- PMDB - maior quantidade de *replies* na região Sudeste e Sul e Norte e devido ao momento atual distribuição mais uniforme sobre todo o território das palavras negativas e positivas.
As palavras que mais se repetem são negativas: ladrão, escárnio e inúteis.

Conclusão

Podemos destacar que cada partido têm uma ligeira área de influência diferenciada por regiões de Brasil então pode-se utilizar esse projeto de forma futura para medir a influência de um usuário popular no Twitter e ver o que as pessoas de cada região do país estão falando sobre ele.

Importante ressaltar que o número de *replies* são influenciados com o momento que passa o país e o partido, caso tenha algum problema ou repercussão na mídia o sentimento pode ser influenciado pelo momento. Então, é interessante manter os textos e as datas armazenadas para ter um registro histórico de como está a evolução a decorrer dos meses e anos do partido.

As dificuldades encontradas:

- Palavras escritas de forma incorreta - Existe uma perda significativa de palavras que estão escritas de forma incorreta, seja por falta de acentuação ou utilização da fonética escrita (exemplo: etaum = então).
- Nem todos os usuários possuem localização ativada - gera uma demanda alta de recuperação de uma base imensa e tempo de recuperação para ter uma amostra significativa.
- Localização escrita de forma incorreta - muitos usuários colocam um apelido, nome coloquial para a cidade ou até mesmo somente o país. Necessitando tratamento e descarte quando somente Brasil ou local não identificado.
- Limitação de pesquisa por quantidade e requisições e tempo da API do Twitter - demanda um tempo de espera bem significativo após esgotar a cota de pesquisas dos textos.

Para superar as dificuldades listadas acima foi necessário utilizar as ferramentas abaixo:

- O dicionário Brazilian Portuguese LIWC 2007 e a biblioteca *PyEnchant* para tratar as palavras escritas de forma incorreta.
- Linguagem de programação Python 3.5 - para recuperação do replies de forma automatizada e configurável para esperar quando o limite de pesquisa na API for atingido.
- A *Google Maps Geocoding API* para trazer a localização de forma que possa ser integrada ao Google Maps e ainda permitir arrumar a descrição do local quando é escrita de uma forma incorreta.

Knime apesar de ser uma ferramenta ótima para gerar a nuvem de palavras, necessita que algumas fontes de dados já estejam bem formatadas, o que demandou um trabalho extra como a exportação do corpus de palavras do dicionário LIWC 2007.

Anexos

Anexo 1 - Recuperar replies

```
def __ajustar_texto(self, texto, flag_lower=False):
    """ Remove caracteres especiais do texto e location
    :param texto:
    :return:
    """
    if texto:
        texto = texto.strip()
        texto = re.sub('((\n)|(\r))', '', texto)
        if flag_lower:
            texto = texto.lower()
        texto = ' '.join(
            re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)|(^A-Za-z \tçéáíóãõúôêâ,;)])(\w+:\V\S+)",
                "", texto).split())
    return texto

def recuperar_replies(self, f):
    """ Recupera os replies e armazena em f em formato csv
    :param f: FileHandler
    :return:
    """
    for partido in self.partidos:
        for m in [9]:
            for d in range(1, 8):
                data = '2017-%02d-%02d' % (m, d)
                print('Replies de %s na data %s ' % (partido, data))
                self.testar_limit_status()
                resultreply = self.api.search(to=partido, since=data, count=100)
                for rreply in resultreply:
                    texto = self.__ajustar_texto(rreply.text.strip(), True)
                    if rreply.user.location and rreply.in_reply_to_status_id and texto:
                        f.write("%s";%s;"%s";"%s";"%s";"%s\n" % (partido,
                                                                    rreply.user.id,
                                                                    self.__ajustar_texto(rreply.user.name),
                                                                    self.__ajustar_texto(rreply.user.location),
                                                                    texto,
                                                                    str(rreply.coordinates if rreply.coordinates else "")))
                    f.flush()
            break
```

Anexo 2 - pré-processamento dos tweets

```
def ajustar_csv_amostra_texto():

    palavras_erradas = []
    if tem_enchant:
        dicionario = enchant.Dict("pt_BR")
    try:
        filename = 'dados/resultado.csv'
        if os.path.exists(filename):
            with open(filename, 'r') as csvfile, open('dados/amostrainicialtexto.csv', 'w') as output:
                reader = csv.DictReader(csvfile, delimiter=';', quotechar='"', lineterminator='\n')
                fieldnames = reader.fieldnames
                writer = csv.DictWriter(output, fieldnames=fieldnames, delimiter=';', quotechar='"',
                                       lineterminator='\n',
                                       quoting=csv.QUOTE_NONNUMERIC)
                writer.writeheader()
                for row in reader:
                    if row['texto'] and re.match('[a-zA-Z]{3,}', row['texto']) and len(row['texto'].split(' ')) > 3:
                        texto_ajustado = ""
                        for s in row['texto'].split(' '):
                            p = re.match('(\w){2,}', s)
                            if p:
                                if row['partido'] == 'PMDB_Nacional' and p.group(0) == 'nacional':
                                    pass
                                else:
                                    if tem_enchant:
                                        if dicionario.check(p.group(0)):
                                            texto_ajustado += ' %s' % p.group(0)
                                        else:
                                            # sugestao = dicionario.suggest(p.group(0))
                                            # if sugestao:
                                            # print(p.group(0), sugestao)
                                            if s not in palavras_erradas:
                                                palavras_erradas.append(s)
                                    else:
                                        texto_ajustado += ' %s' % p.group(0)

                        if texto_ajustado:
                            writer.writerow({'partido': row['partido'],
                                             'userid': row['userid'],
                                             'username': row['username'],
                                             'location': row['location'],
                                             'texto': texto_ajustado,
                                             'coordinates': row['coordinates']
                                             })
                    # if palavras_erradas:
                    # print(palavras_erradas)

    except Exception as e:
        print('ERRO: %s' % str(e))
```

Anexo 3 - extração das palavras positivas e negativas

```
def extrair_posemo_negemo():
    palavras_sent = {}

    for p in dicionario:
        if '*' not in p:
            if 'posemo' in dicionario[p]:
                palavras_sent[p] = 'pos'
            if 'negemo' in dicionario[p]:
                palavras_sent[p] = 'neg'

    if palavras_sent:
        od = collections.OrderedDict(sorted(palavras_sent.items()))
        writer_pos = csv.DictWriter(open('dados/corpus_positivo_negativo_from_LIWC2007_.csv', 'w'),
                                    fieldnames=['palavra', 'sentimento'],
                                    delimiter=';', quotechar='"',
                                    lineterminator='\n',
                                    quoting=csv.QUOTE_NONNUMERIC)
        writer_pos.writeheader()
        for k, v in od.items():
            writer_pos.writerow({'palavra': k, 'sentimento': v})

    return None
```

Anexo 4 - Resultado API geocode

```
{
  "results" : [
    {
      "address_components" : [
        {
          "long_name" : "São Paulo",
          "short_name" : "São Paulo",
          "types" : [ "locality", "political" ]
        },
        {
          "long_name" : "São Paulo",
          "short_name" : "São Paulo",
          "types" : [ "administrative_area_level_2", "political" ]
        },
        {
          "long_name" : "São Paulo",
          "short_name" : "SP",
          "types" : [ "administrative_area_level_1", "political" ]
        },
        {
          "long_name" : "Brasil",
          "short_name" : "BR",
          "types" : [ "country", "political" ]
        }
      ],
      "formatted_address" : "São Paulo, SP, Brasil",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : -23.3566039,
            "lng" : -46.3650844
```

```

    },
    "southwest" : {
        "lat" : -24.0082209,
        "lng" : -46.825514
    },
    "location" : {
        "lat" : -23.5505199,
        "lng" : -46.63330939999999
    },
    "location_type" : "APPROXIMATE",
    "viewport" : {
        "northeast" : {
            "lat" : -23.3566039,
            "lng" : -46.3650844
        },
        "southwest" : {
            "lat" : -24.0082209,
            "lng" : -46.825514
        }
    },
    "place_id" : "ChIJ0WGkg4FEzpQRrlsz_whLqZs",
    "types" : [ "locality", "political" ]
},
"status" : "OK"
}

```

Anexo 5 - Separar texto e localização

```
def separar_texto_location():
```

```
    location_list = []
```

```
    with open('dados/amostrainicialtexto.csv', 'r', encoding='utf-8') as input:
```

```
        reader = csv.DictReader(input, delimiter=';', quotechar='"', lineterminator='\n')
```

```
        for row in reader:
```

```
            if row['texto'] and ((row['location'] and re.match('(\w{2,})+', row['location']) and
                                not row['location'].startswith('Brasil') and
                                row['location'] != 'br')):
```

```
                # print(row['location'], row['texto'])
```

```
                if row['location'] not in location_list:
```

```
                    location_list.append(row['location'])
```

```
if location_list:
```

```
    with open('dados/localidades.csv', 'w', encoding='utf-8') as output:
```

```
        writer = csv.DictWriter(output, fieldnames=['location', 'formatted_address', 'lat', 'lng', 'lat_lng'],
                                delimiter=';', quotechar='"',
                                lineterminator='\n',
                                quoting=csv.QUOTE_NONNUMERIC)
```

```
        writer.writeheader()
```

```
        for location in location_list:
```

```
            print('Recuperando: %s' % location)
```

```
            url = "https://maps.googleapis.com/maps/api/geocode/json?address=%s" % location
```

```
            r = requests.get(url)
```

```
            if r.status_code == 200:
```

```
                j = loads(r.content.decode())
```

```

        if j.get('results') and not j['results'][0]['formatted_address'].startswith(
            'Brazil') and 'Brazil' in j['results'][0]['formatted_address']:
            writer.writerow({'location': location,
                            'formatted_address': j['results'][0]['formatted_address'],
                            'lat': j['results'][0]['geometry']['location']['lat'],
                            'lng': j['results'][0]['geometry']['location']['lng'],
                            'lat_lng': '%s,%s' % (j['results'][0]['geometry']['location']['lat'],
                                                    j['results'][0]['geometry']['location']['lng']),
                            })
    else:
        print('ERRO: %s' % r.status_code)

```

Anexo 6 - Separar palavras por localidade

```

def separar_palavras_localidade():
    location_list = {}
    with open('dados/localidades.csv', 'r', encoding='utf-8') as input:
        reader = csv.DictReader(input, delimiter=',', quotechar='"', lineterminator='\n')
        for row in reader:
            location_list[row['location']] = {'formatted_address': row['formatted_address'],
                                              'lat': row['lat'],
                                              'lng': row['lng'],
                                              'lat_lng': row['lat_lng']}

    palavras_positivas = []
    palavras_negativas = []

    flag_corpus = 1

    if flag_corpus == 0:
        with open('dados/corpus-traduzido-positivo.csv', 'r', encoding='utf-8') as input:
            reader = csv.DictReader(input, delimiter=',', quotechar='"', lineterminator='\n',
                                    fieldnames=['eng', 'pt'])
            for row in reader:
                palavras_positivas.append(row['pt'])

        with open('dados/corpus-traduzido-negativo.csv', 'r', encoding='utf-8') as input:
            reader = csv.DictReader(input, delimiter=',', quotechar='"', lineterminator='\n',
                                    fieldnames=['eng', 'pt'])
            for row in reader:
                palavras_negativas.append(row['pt'])
        path_palavras_location = 'dados/palavras_location_separadas.csv'
    else:
        path_palavras_location = 'dados/palavras_location_separadas2.csv'
        reader_corpus = csv.DictReader(open('dados/corpus_positivo_negativo_from_LIWC2007_.csv', 'r'),
                                        fieldnames=['palavra', 'sentimento'],

```

```

        delimiter=';', quotechar='"',
        lineterminator='\n',
        quoting=csv.QUOTE_NONNUMERIC)
for row in reader_corpus:
    if row['sentimento'] == 'pos':
        palavras_positivas.append(row['palavra'])
    elif row['sentimento'] == 'neg':
        palavras_negativas.append(row['palavra'])

palavra_location = {}
tweets_per_user = {}
with open('dados/amostrainicialtexto.csv', 'r', encoding='utf-8') as input:
    reader = csv.DictReader(input, delimiter=';', quotechar='"', lineterminator='\n')
    for row in reader:
        local = location_list.get(row['location'])
        if local:
            # Verifica se repete mesmo tweet para o mesmo usuario e partido, se repetir nao e importante
            user_partido = '%s_%s' % (row['partido'], row['userid'])
            if tweets_per_user.get(user_partido) is None:
                tweets_per_user[user_partido] = []
            if row['texto'] in tweets_per_user[user_partido]:
                continue
            else:
                tweets_per_user[user_partido].append(row['texto'])
                for palavra in row['texto'].split():
                    if palavra_location.get(row['partido']) is None:
                        palavra_location[row['partido']] = {}
                    if palavra not in stopwords and palavra_location[row['partido']].get(palavra) is None:
                        if palavra in palavras_positivas:
                            palavra_location[row['partido']][palavra] = {'location': []}
                            palavra_location[row['partido']][palavra]['tipo'] = 'pos'
                        elif palavra in palavras_negativas:
                            palavra_location[row['partido']][palavra] = {'location': []}
                            palavra_location[row['partido']][palavra]['tipo'] = 'neg'
                        if palavra_location[row['partido']].get(palavra) and
palavra_location[row['partido']][palavra].get('tipo'):
                            palavra_location[row['partido']][palavra]['location'].append(local)

with open(path_palavras_location, 'w', encoding='utf-8') as output:
    writer = csv.DictWriter(output, fieldnames=['partido', 'palavra', 'tipo', 'lat', 'lng'],
        delimiter=';', quotechar='"',
        lineterminator='\n',
        quoting=csv.QUOTE_NONNUMERIC)
    writer.writeheader()
    for partido in palavra_location:
        for palavra in palavra_location[partido]:
            for local in palavra_location[partido][palavra]['location']:
                writer.writerow({'partido': partido,

```

```

        'palavra': palavra,
        'tipo': palavra_location[partido][palavra]['tipo'],
        'lat': local['lat'],
        'lng': local['lng']
    })

```

Anexo 7 - Utilização da biblioteca Flask Google Maps - usuários únicos e distribuídos por partido

```

@app.route("/")
def mapview():
    locationsrede45 = []
    locationspt = []
    locationspmdb = []

    with open('resultado2a.csv', 'r', encoding="utf8") as csvfile:
        reader = csv.DictReader(csvfile, delimiter=';', quotechar='"', lineterminator='\n')
        try:
            for row in reader:
                if row['formatted_address'] and not row['formatted_address'].startswith('Brazil'):

                    if row['partido'] == 'Rede45':
                        locationsrede45.append({'lat': row['lat'], 'lng': row['lng'], 'infobox': row['formatted_address'], 'icon':
'http://maps.google.com/mapfiles/ms/icons/blue-dot.png'})
                    elif row['partido'] == 'ptbrasil':
                        locationspt.append({'lat': row['lat'], 'lng': row['lng'], 'infobox': row['formatted_address'], 'icon':
'http://maps.google.com/mapfiles/ms/icons/red-dot.png'})
                    else:
                        locationspmdb.append({'lat': row['lat'], 'lng': row['lng'], 'infobox': row['formatted_address'], 'icon':
'http://maps.google.com/mapfiles/ms/icons/green-dot.png'})

            except Exception as e:
                print('ERRO %s' % str(e))

    maparede45 = Map(
        identifier="view-maparede45",
        varname="mapa",
        style="height:600px;width:1024px;margin:0;",
        lat=-14.235004,
        lng=-51.92528,
        zoom=5,
        markers=locationsrede45
    )

    mapapt = Map(
        identifier="view-mapapt",
        varname="mapa",
        style="height:600px;width:1024px;margin:0;",

```

```

lat=-14.235004,
lng=-51.92528,
zoom=5,
markers=locationspt
)

mapapmdb = Map(
    identifier="view-mapapmdb",
    varname="mapa",
    style="height:600px;width:1024px;margin:0;",
    lat=-14.235004,
    lng=-51.92528,
    zoom=5,
    markers=locationspmdb
)

return render_template('template.html', maparede45=maparede45, mapapt=mapapt,
mapapmdb=mapapmdb)

```

Anexo 8 - Utilização da biblioteca Flask Google Maps - localização das palavras

```

@app.route("/palavras")
def palavras():
    locationsrede45 = []
    locationsrede45neg = []
    locationspt = []
    locationsptneg = []
    locationspmdb = []
    locationspmdbneg = []
    qtd_pos_rede45 = qtd_neg_rede45 = 0
    qtd_pos_pt = qtd_neg_pt = 0
    qtd_pos_pmdb = qtd_neg_pmdb = 0

    with open('palavras_location_separadas2.csv', 'r', encoding="utf8") as csvfile:
        reader = csv.DictReader(csvfile, delimiter=';', quotechar="'", lineterminator='\n')
        try:
            for row in reader:

                if row['tipo'] == 'pos':
                    icone = 'http://maps.google.com/mapfiles/ms/icons/blue-dot.png'
                else:
                    icone = 'http://maps.google.com/mapfiles/ms/icons/red-dot.png'

                if row['partido'] == 'Rede45':

                    if row['tipo'] == 'pos':
                        locationsrede45.append({'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'],
                                                'icon': icone})
                        qtd_pos_rede45 += 1

```



```

else:
    locationsrede45neg.append({'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'],
                              'icon': icone})
    qtd_neg_rede45 += 1

elif row['partido'] == 'ptbrasil':

    if row['tipo'] == 'pos':
        locationspt.append(
            {'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'], 'icon': icone})
        qtd_pos_pt += 1
    else:
        locationsptneg.append(
            {'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'], 'icon': icone})
        qtd_neg_pt += 1
else:
    if row['tipo'] == 'pos':
        locationspmdb.append(
            {'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'], 'icon': icone})
        qtd_pos_pmdb += 1
    else:
        locationspmdbneg.append(
            {'lat': row['lat'], 'lng': row['lng'], 'infobox': row['palavra'], 'icon': icone})
        qtd_neg_pmdb += 1

```

```

except Exception as e:
    print('ERRO %s' % str(e))

```

```

maparede45 = Map(
    identifier="view-maparede45",
    varname="maparede45",
    style="height:600px;width:1024px;margin:0;",
    lat=-14.235004,
    lng=-51.92528,
    zoom=4,
    markers=locationsrede45,
    cluster=False
)

```

```

maparede45neg = Map(
    identifier="view-maparede45neg",
    varname="maparede45",
    style="height:600px;width:1024px;margin:0;",
    lat=-14.235004,
    lng=-51.92528,
    zoom=4,
    markers=locationsrede45neg,
    cluster=False
)

```

```

maparedept = Map(
    identifier="view-mapapt",
    varname="maparede45",

```

```

        style="height:600px;width:1024px;margin:0;",
        lat=-14.235004,
        lng=-51.92528,
        zoom=4,
        markers=locationspt
    )
    maparedeptneg = Map(
        identifier="view-mapaptneg",
        varname="maparede45",
        style="height:600px;width:1024px;margin:0;",
        lat=-14.235004,
        lng=-51.92528,
        zoom=4,
        markers=locationsptneg
    )

    maparedepmdb = Map(
        identifier="view-mapapmdb",
        varname="maparede45",
        style="height:600px;width:1024px;margin:0;",
        lat=-14.235004,
        lng=-51.92528,
        zoom=4,
        markers=locationspmdb
    )

    maparedepmdbneg = Map(
        identifier="view-mapapmdbneg",
        varname="maparede45",
        style="height:600px;width:1024px;margin:0;",
        lat=-14.235004,
        lng=-51.92528,
        zoom=4,
        markers=locationspmdbneg
    )

    return render_template('palavras.html',
        maparede45=maparede45, maparede45neg=maparede45neg,
        maparedept=maparedept, maparedeptneg=maparedeptneg,
        maparedepmdb=maparedepmdb, maparedepmdbneg=maparedepmdbneg,
        qtd_pos_rede45=qtd_pos_rede45, qtd_neg_rede45=qtd_neg_rede45,
        qtd_pos_pt=qtd_pos_pt, qtd_neg_pt=qtd_neg_pt,
        qtd_pos_pmdb=qtd_pos_pmdb, qtd_neg_pmdb=qtd_neg_pmdb)

```