

Capítulo

2

Extração e Classificação de Dados Semânticos do Twitter

Clarissa Castellã Xavier¹ e Marlo Souza²

¹Universidade Federal do Rio Grande do Sul (UFRGS)

²Universidade Federal da Bahia (UFBA)

Abstract

Twitter is a social network and microblogging service in which registered users read and post messages called tweets. Tweets have a maximum of 280 characters and cover every conceivable subject, from simple activity updates and news coverage to opinions about arbitrary topics. In this way, Twitter emerges as a valuable data source to get information about what people think and feel about the most different subjects. In this context, this course presents different approaches for extracting and processing information from Twitter using Natural Language Processing (NLP) and Machine Learning techniques, examining tools and methods to collect and analyze semantic information from tweets.

Resumo

O Twitter é uma rede social e serviço de microblog onde usuários registrados leem e postam mensagens chamadas tweets. Os tweets podem ter no máximo 280 caracteres e abrangem todos os assuntos concebíveis, desde simples atualizações sobre atividades e notícias jornalísticas até opiniões sobre tópicos arbitrários. Desta forma, o Twitter surge como uma valiosa fonte de dados para obtenção de informações sobre o que as pessoas pensam e sentem sobre os mais diversos assuntos. Neste contexto, este curso apresenta diferentes abordagens para extração e processamento de informações do Twitter usando técnicas de Processamento de Linguagem Natural (PLN) e Aprendizado de Máquina, examinando ferramentas e métodos para coleta e análise de informações semânticas dos tweets.

2.1. Introdução

É clara a posição privilegiada que as mídias sociais têm na vida das pessoas, particularmente no contexto brasileiro. Portanto, entender o comportamento dos seus usuários

é um passo importante para qualquer organização. Toda esta informação é, no entanto, difícil de gerir, sendo que esta dificuldade deriva principalmente do fato de que estes dados se encontram em formato não-estruturado ou semiestruturado, como texto, imagens e páginas da web.

O Twitter ¹ é um serviço de microblog e rede social lançado no final de 2006, no qual usuários postam mensagens de até 280 caracteres. No primeiro trimestre de 2018 o serviço teve uma média de 336 milhões usuários mensais ativos ². O Twitter é caracterizado como uma mídia informativa para o público brasileiro. De acordo com Recuero [da Cunha Recuero 2003], no contexto brasileiro, os usuários se envolvem com a plataforma mais como um meio de coletar e transmitir informações do que de se envolver em interações sociais, como conversas. De acordo com esse estudo, cerca de 62% dos *tweets* têm conteúdo informativo, enquanto cerca de 48% são de natureza conversacional, com 10% de textos com ambas as características. Em relação aos textos com perfil informativo, cerca de 25% possuem conteúdo opinativo, ou seja, o usuário expressa opiniões ou sentimentos. De fato, de acordo com um estudo recente sobre o perfil de acesso a notícias no mundo [Newman et al. 2016], 72% dos brasileiros residentes em áreas urbanas usam mídias sociais como fonte de notícias, dos quais 13% utilizam o Twitter como principal rede social para esse fim (15% na população abaixo de 35 anos).

É evidente que a quantidade de informações disponíveis cresceu significativamente devido aos novos meios de indexação de informações e novos recursos de distribuição. Devido à esta grande quantidade de dados, no entanto, não é fácil, nem mesmo gerenciável, encontrar e explorar informações que sejam estratégicas ou mesmo relevantes para um determinado indivíduo ou organização. A principal razão para essa dificuldade está na natureza não estruturada dos dados, já que seu tratamento computacional não é trivial. Neste contexto as tecnologias de informação e, em particular, a Análise de Textos têm um papel crucial em facilitar a obtenção e o processamento de dados relevantes na Web.

A Análise de Textos estuda como aplicar métodos de Inteligência Computacional para extrair automaticamente informações estruturadas de documentos não estruturados [Dale 2008]. Para Dale [Dale 2008], esta área visa desenvolver soluções para oito grandes problemas: Recuperação de Informação, Categorização e Agrupamento de Textos, Reconhecimento de Entidades, Co-Referência Nominal, Sumarização de Textos, Extração de Informação, Análise de Sentimentos (Polaridade) e Sistemas de Perguntas e respostas.

Neste trabalho vamos nos concentrar em dois problemas que acreditamos serem relevantes para lidar com os microtextos do Twitter: Classificação de Polaridade e Reconhecimento de Entidades.

A classificação de polaridade é uma tarefa bem conhecida de PLN. Dado um texto, seu objetivo é identificar se existe um conteúdo subjetivo no mesmo e obter a polaridade do sentimento transmitido pela informação, por exemplo, se o texto expressa um sentimento positivo, negativo ou neutro. Dadas as limitações de tamanho dos *tweets*, a classificação de sentimentos das mensagens do Twitter é normalmente executada no nível da

¹<https://twitter.com/>

²<https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/>

sentença; no entanto, a linguagem informal e especializada faz com que esta seja uma tarefa singular.

A extração de entidades (EE) (também conhecida como Reconhecimento de Entidades Nomeadas (REN)) é uma sub-tarefa da área de extração de informações. Seu objetivo é localizar e classificar entidades nomeadas no texto. A EE costuma ser uma das primeiras etapas do *pipeline* de extração de informações. O estilo breve, informal e ruidoso do Twitter apresenta desafios. Estudaremos diferentes abordagens para detectar as entidades citadas em um *Tweet*.

Neste curso apresentamos diferentes técnicas de extração e classificação de dados semânticos para obtenção de informações do Twitter. Também falamos sobre as ferramentas que implementam essas técnicas aprendendo a usar esses recursos na prática. Aplicaremos estas habilidades em um estudo de caso, a fim de praticar importantes habilidades de manipulação de dados, aprendendo como diferentes formas de análise de dados podem ser usadas.

2.2. Extraindo dados do Twitter

O Twitter é uma ferramenta de *microblogging* lançada no final de 2006 na qual os usuários postam mensagens de até 280 caracteres chamadas *tweets*. Apenas usuários registrados podem postar *tweets*, mas aqueles que não estão registrados podem lê-los. O Twitter pode ser acessado através de sua interface no *website*, por meio do Serviço de Mensagens Curtas (SMS) ou através do aplicativo para dispositivos móveis ("app").

É possível coletar informações do Twitter utilizando a API pública ³, bem como utilizando aplicativos e bibliotecas alternativas. A seguir veremos o funcionamento da API do Twitter e da ferramenta Twint ⁴ e como utilizá-las para extrair dados.

2.2.1. API do Twitter

Os *tweets* estão gratuitamente disponíveis para desenvolvedores de *software* através de APIs públicas⁵. Essas APIs podem ser classificadas em dois tipos [Kumar et al. 2013]:

1. REST APIs: usam a estratégia *pull* para recuperação de dados, ou seja, para coletar informações o usuário deve explicitamente fazer uma solicitação.
2. *Streaming* APIs: usam a estratégia *push* para recuperação de dados, ou seja, depois que uma solicitação de informações é feita, a API fornece um fluxo contínuo de atualizações sem necessitar nenhuma outra solicitação.

As APIs do Twitter incluem uma ampla variedade de terminais que se dividem em cinco grupos principais: Contas e usuários, Mensagens diretas, Anúncios, Ferramentas de *publisher* e SDKs, *Tweets* e respostas [Twitter 2018]. Neste curso iremos nos concentrar no último grupo.

³<https://developer.twitter.com/>

⁴<https://github.com/twintproject/twint>

⁵<https://developer.twitter.com/>

O grupo de APIs *Tweets* e Respostas torna os *tweets* e as respostas públicas disponíveis para os desenvolvedores e permite que estes também postem *tweets*. Os desenvolvedores podem acessar os *tweets* pesquisando por palavras-chave específicas ou solicitando conteúdo de contas específicas [Twitter 2018].

2.2.1.1. Acessar a API

Para acessar a API do Twitter é necessário registrar uma aplicação. Por padrão os aplicativos podem acessar apenas informações públicas. Algumas aplicações, como por exemplo aquelas responsáveis pelo envio ou recebimento de mensagens diretas, exigem permissões adicionais [Twitter 2018].

Para obter este acesso é necessário obter uma chave e um *token* de acesso da API, um *token* e uma chave consumidora. Esse procedimento é feito em <http://dev.twitter.com/apps>.

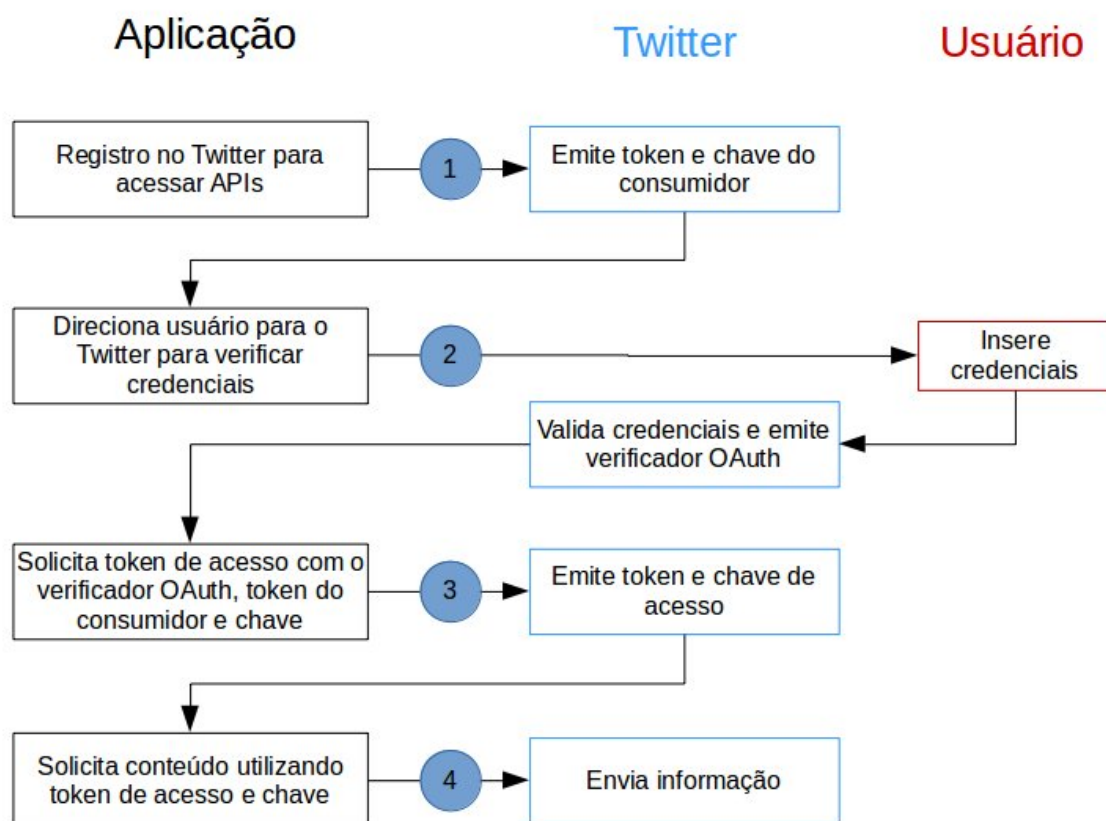


Figura. 2.1. Fluxo OAuth para obtenção do token de acesso à API do Twitter - adaptado de [Kumar et al. 2013]

A autenticação das requisições à API são realizadas utilizando autenticação aberta (OAuth). A Figura 2.1 sumariza os passos envolvidos na autenticação. A API somente pode ser acessada por aplicações seguindo os passos abaixo [Kumar et al. 2013]:

1. As aplicações (consumidores) precisam se registrar no Twitter (em `http://dev.twitter.com`). Neste processo a aplicação recebe uma chave e um *token* que o aplicativo deve usar para se autenticar.
2. A aplicação usa a chave e o *token* para criar um link exclusivo do Twitter para o qual o usuário é direcionado para autenticação. O usuário autoriza a aplicação autenticando-se no Twitter. O Twitter verifica a identidade do usuário e fornece um verificador OAuth (PIN).
3. O usuário informa o PIN para o aplicativo. O aplicativo usa o PIN para solicitar um *token* e uma chave de acesso exclusivos para o usuário.
4. Utilizando o *token* e a chave de acesso o aplicativo autentica o usuário no Twitter e chama a API em nome do usuário.

O *token* e a chave de acesso do usuário não podem ser alterados e podem ser armazenados em cache pelo aplicativo para futuras solicitações. Desta forma, este processo só precisa ser executado uma vez.

O código Python reproduzido na Listagem 2.1 implementa o fluxo OAuth para acesso à API do Twitter ilustrado na Figura 2.1. Como podemos ver, foi utilizada a biblioteca *oauth2*⁶. Ela lida com todas as etapas do protocolo OAuth 2.0 necessárias para fazer chamadas de API.

Listagem 2.1. Código em Python para acessar API do Twitter

```
import oauth2
def oauth_req(url, CHAVE_ACESSO, TOKEN_ACESSO, http_method=
    "GET", post_body="", http_headers=None):
    token = oauth2.Token(key=CHAVE_ACESSO, secret=
        TOKEN_ACESSO)
    consumo = oauth2.Consumer(key=CHAVE_CONSUMO, secret=
        TOKEN_CONSUMO)
    cliente = oauth2.Client(consumo, token)
    resp, conteudo = cliente.request( url, method=
        http_method, body=post_body, headers=http_headers )
    return conteudo
```

2.2.2. Coletar dados sem acessar a API

Existem outras abordagens para coletar dados do Twitter além da API. Uma opção é a ferramenta Twint implementada em Python. Ele utiliza os operadores de pesquisa do Twitter para capturar *tweets* de usuários específicos, relacionados a determinados tópicos, *hashtags* e tendências. Também faz consultas especiais ao Twitter, permitindo obter os seguidores de um usuário, *tweets* curtidos e seguidores.

A seguir veremos exemplos do uso da API e do Twint para extrair os seguintes tipos de informação:

⁶<https://pypi.org/project/python-oauth2/>

- Informações sobre um usuário
- Seguidores de um usuário
- Quem o usuário segue
- *Tweets* publicados
- Resultados de uma pesquisa

2.2.3. Exemplos Práticos

A seguir veremos exemplos de como coletar informações do Twitter utilizando a API pública e a biblioteca Python Twint. Abordaremos as seguintes tarefas: obter informações sobre um usuário, obter seguidores de um usuário, obter quem o usuário segue, obter *tweets* e obter resultados de uma pesquisa

2.2.3.1. Obter informações sobre um usuário

Via API

A API principal do Twitter é responsável pela manipulação e consulta dos dados. Qualquer método dessa API é precedido da URI <http://api.twitter.com/version/>, sendo que *version* é a versão da API (atualmente 1).

Cada usuário do Twitter está associado a um identificador, também chamado de nome de tela (*screen_name*), e um ID (*user_id*).

O método *users/show*⁷ retorna as informações do perfil do usuário. Ela aceita um nome de usuário válido como parâmetro e retorna o perfil deste usuário no Twitter.

A Listagem 2.2 mostra o código Python para obter os dados de um perfil.

Listagem 2.2. Código em Python para chamada API *users/show*.

```
def info_usr(usuario):
    GET_USR_URL = "https://api.twitter.com/1.1/users/show.
        json?screen_name="+usuario
    req = oauth_req(GET_USR_URL, TOKEN_ACESSO, CHAVE_ACESSO)
    return req
```

Um exemplo de chamada do método criado no código 2.2 seria: `info_usr('twitterbrasil')`. Um objeto de usuário típico é formatado como na Listagem 2.3.

Listagem 2.3. Parte do objeto JSON retornado pelo método *info_usr('twitterbrasil')*.

```
{...
    "created_at": "Thu_Mar_10_22:54:23_+0000_2011",
```

⁷<https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-show.html>

```

"description": "Bem-vindos_\u00e0_conta_oficial_do_
    Twitter_Brasil!_Precisa_de_ajuda?_Acesse_https://t.co
    /Nu5ZS0w4UD",
"favourites_count": 1537,
"followers_count": 2181337,
"id": 263884490,
"lang": "pt",
"location": "Brasil",
"name": "Twitter_Brasil",
"screen_name": "TwitterBrasil",
"statuses_count": 7434,
"time_zone": null,
"translator_type": "regular",
"url": "http://t.co/GuzHOnaY84",
"verified": true
...}

```

A ferramenta Twint não disponibiliza esta funcionalidade.

2.2.3.2. Obter seguidores de um usuário

Via API

O método *followers/list*⁸ retorna uma coleção de objetos de usuário contendo os usuários que seguem o perfil informado como parâmetro. Os resultados são fornecidos em grupos de 20 usuários e várias páginas de resultados podem ser navegadas usando o valor *next_cursor* em solicitações subsequentes.

A Listagem 2.4 mostra o código Python para obter seguidores de um perfil.

Listagem 2.4. Código em Python para chamada API *followers/list*.

```

def seguidores_usr(usuario):
    GET_USR_URL = "https://api.twitter.com/1.1/followers/
        list.json?cursor=-1&skip_status=true&
        include_user_entities=false&screen_name="+usuario
    req = oauth_req(GET_USR_URL, TOKEN_ACESSO, CHAVE_ACESSO)
    return req

```

Via Twint

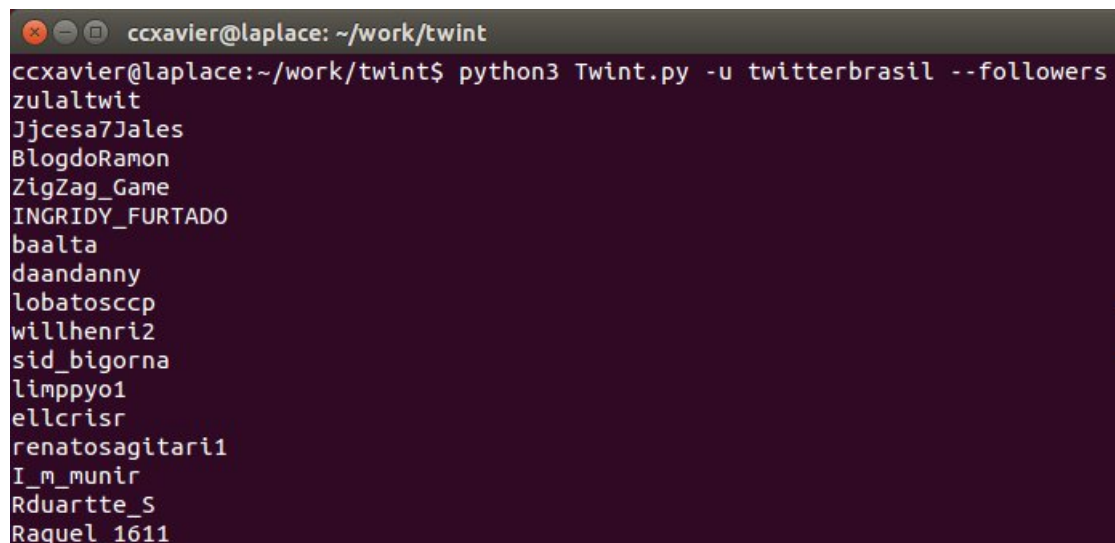
O comando a seguir lista os usuários que seguem uma determinada conta:

```
python3 Twint.py -u [usuário] -followers.
```

A Figura 2.2 mostra a chamada e o retorno do comando

```
python3 Twint.py -u twitterbrasil -followers.
```

⁸<https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-list.html>

A terminal window with a dark background and light text. The prompt is 'ccxavier@laplace: ~/work/twint'. The command entered is 'python3 Twint.py -u twitterbrasil --followers'. The output is a list of usernames: zulaltwit, Jjcesa7Jales, BlogdoRamon, ZigZag_Game, INGRIDY_FURTADO, baalta, daandanny, lobatosccp, willhenri2, sid_bigorna, limppy01, ellcrisr, renatosagitari1, I_m_munir, Rduartte_5, and Raquel_1611.

```
ccxavier@laplace: ~/work/twint
ccxavier@laplace:~/work/twint$ python3 Twint.py -u twitterbrasil --followers
zulaltwit
Jjcesa7Jales
BlogdoRamon
ZigZag_Game
INGRIDY_FURTADO
baalta
daandanny
lobatosccp
willhenri2
sid_bigorna
limppy01
ellcrisr
renatosagitari1
I_m_munir
Rduartte_5
Raquel_1611
```

Figura. 2.2. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil --followers`

2.2.3.3. Obter quem o usuário segue

Via API

O método *friends/list*⁹ retorna uma coleção de objetos de usuário contendo os usuários seguidos pelo perfil informado como parâmetro. Os resultados são fornecidos em grupos de 20 usuários e várias páginas de resultados podem ser navegadas usando o valor *next_cursor* em solicitações subsequentes.

A Listagem 2.5 mostra o código Python para obter os amigos de um perfil.

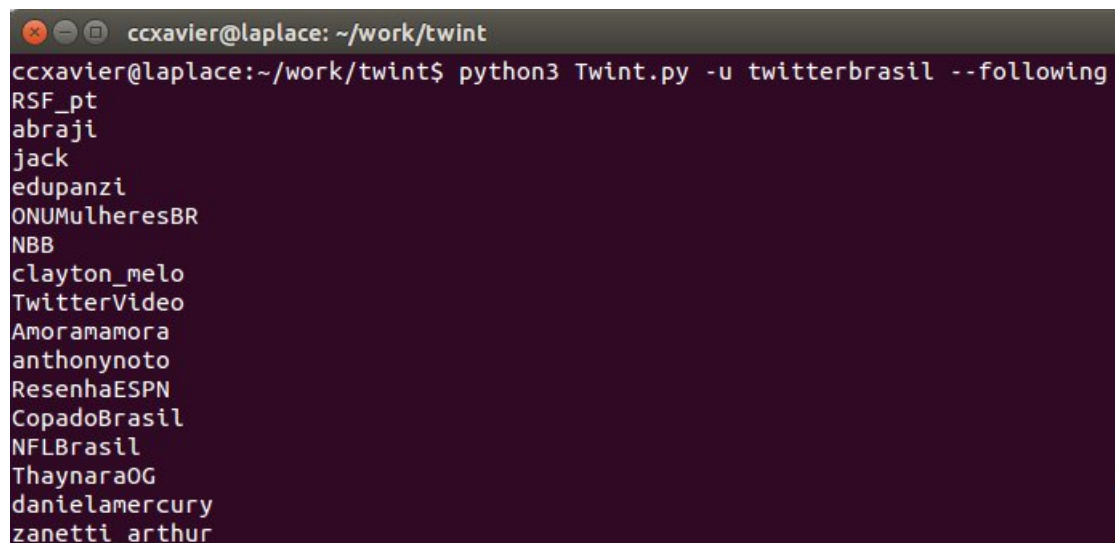
Listagem 2.5. Código em Python para chamada API *friends/list*.

```
def seguidos_usr(usuario):
    GET_USR_URL = "https://api.twitter.com/1.1/friends/list.
        json?cursor=-1&skip_status=true&include_user_entities
        =false&screen_name="+usuario
    req = oauth_req(GET_USR_URL, TOKEN_ACESSO, CHAVE_ACESSO)
    return req
```

Via Twint

O comando `python3 Twint.py -u [usuário] --following` retorna a lista os usuários que seguem a conta do usuário informado como parâmetro. A Figura 2.3 mostra a chamada e o retorno do comando `python3 Twint.py -u twitterbrasil --following`.

⁹<https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-list>

A terminal window with a dark background and light text. The window title is 'ccxavier@laplace: ~/work/twint'. The command executed is 'python3 Twint.py -u twitterbrasil --following'. The output is a list of usernames: RSF_pt, abraji, jack, edupanzi, ONUMulheresBR, NBB, clayton_melo, TwitterVideo, Amoramamora, anthonymoto, ResenhaESPN, CopadoBrasil, NFLBrasil, ThaynaraOG, danielamercury, and zanetti_arthur.

```
ccxavier@laplace: ~/work/twint
ccxavier@laplace:~/work/twint$ python3 Twint.py -u twitterbrasil --following
RSF_pt
abraji
jack
edupanzi
ONUMulheresBR
NBB
clayton_melo
TwitterVideo
Amoramamora
anthonymoto
ResenhaESPN
CopadoBrasil
NFLBrasil
ThaynaraOG
danielamercury
zanetti_arthur
```

Figura. 2.3. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil --following`

2.2.3.4. Obter *Tweets*

Via API

Um *Tweet* pode ser de autoria do usuário ou um tipo especial de *Tweet* chamado *Retweet*, criado quando um usuário reposta o *Tweet* elaborado por outro usuário. Os *tweets* de um usuário podem ser recuperados usando o REST e a API de *streaming*.

O método `statuses/usertimeline`¹⁰ retorna uma coleção dos *tweets* mais recentes postados por um usuário utilizando REST API. As linhas de tempo pertencentes a usuários protegidos só podem ser solicitadas quando o usuário autenticado "possui" a linha do tempo ou é um seguidor aprovado.

Esse método retorna até 3.200 dos *tweets* mais recentes de um usuário. Os resultados são fornecidos em grupos e várias páginas de resultados podem ser navegadas usando o valor `next_cursor` em solicitações subsequentes. Cada página retorna até 200 *tweets*. O parâmetro `max_id` é usado para paginar. Para recuperar a próxima página, usamos o ID do *Tweet* mais antigo na lista como o valor desse parâmetro na solicitação subsequente. Em seguida, a API recuperará apenas os *tweets* cujos IDs estão abaixo do valor fornecido.

A Listagem 2.6 mostra o código Python para obter o último *Tweet* de um perfil.

Listagem 2.6. Código em Python para chamada API `statuses/usertimeline`.

```
def tweets_usr(usuario):
    GET_USR_URL = "https://api.twitter.com/1.1/statuses/
        user_timeline.json?count=1&screen_name="+usuario
    req = oauth_req(GET_USR_URL, TOKEN_ACESSO, CHAVE_ACESSO)
    return req
```

¹⁰https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html

O método *statuses/filter*¹¹ retorna uma coleção dos *tweets* que correspondam a um ou mais parâmetros de filtro usando *Streaming*. Ele permite que o cliente use uma única conexão, persistindo a conexão com a API.

A Listagem 2.7 mostra o código Python para criar uma solicitação POST para a API e buscar os resultados da pesquisa. O parâmetro é o termo que será sendo seguido. Por exemplo, se o parâmetro *termo* for *twitter*, o método irá imprimir os *tweets* contendo este termo sendo criados publicamente na plataforma. O código chama as bibliotecas auxiliares *oauth2*, *json*¹² e *urllib2*¹³.

Listagem 2.7. Código em Python para chamada API *statuses/filter*.

```
def segue_tweets(termo):
    url = "https://stream.twitter.com/1.1/statuses/filter.
        json?track="+termo
    http_method="POST"
    post_body=""
    http_headers=None
    token = oauth2.Token(key=CHAVE_ACESSO, secret=
        TOKEN_ACESSO)
    consumo = oauth2.Consumer(key=CHAVE_CONSUMO, secret=
        TOKEN_CONSUMO)
    cliente = oauth2.Client(consumo, token)
    headers = {}
    req = oauth2.Request.from_consumer_and_token(
        cliente.consumer, token=cliente.token,
        http_method="POST", http_url=url)
    req.sign_request(cliente.method, cliente.consumer,
        cliente.token)
    headers.update(req.to_header())
    body = req.to_postdata()
    headers['Content-Type'] = 'application/x-www-form-
        urlencoded'
    req = urllib2.Request(url, body, headers=headers)
    try:
        f = urllib2.urlopen(req)
    except urllib2.HTTPError, e:
        data = e.fp.read(1024)
        raise Exception(e, data)
    for line in f:
        d = json.loads(line)
        try:
            print d["user"]["name"], d["text"]
        except:
```

¹¹https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html

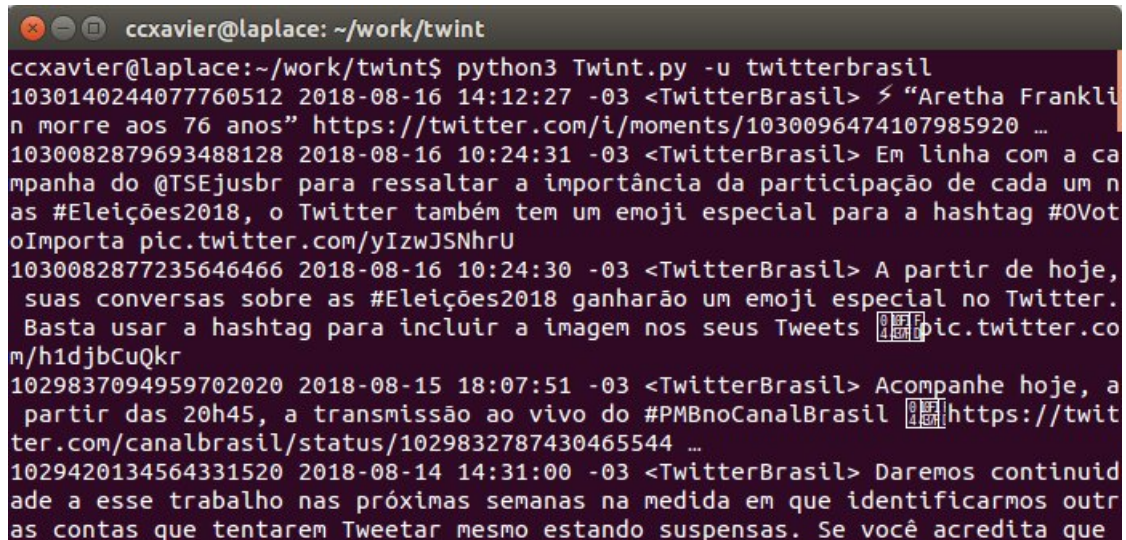
¹²<https://docs.python.org/2/library/json.html>

¹³<https://docs.python.org/2/library/urllib2.html>

```
print d.get("id")
```

Via Twint

O comando a seguir retorna todos os *Tweets* da *timeline* do usuário informado como parâmetro: `python3 Twint.py -u [usuário]`. A Figura 2.4 mostra a chamada e o retorno do comando `python3 Twint.py -u twitterbrasil`.



```
ccxavier@laplace: ~/work/twint
ccxavier@laplace:~/work/twint$ python3 Twint.py -u twitterbrasil
1030140244077760512 2018-08-16 14:12:27 -03 <TwitterBrasil> <img alt="Twitter bird icon" data-bbox="285 290 305 305"/> "Aretha Franklin morre aos 76 anos" https://twitter.com/i/moments/1030096474107985920 ...
1030082879693488128 2018-08-16 10:24:31 -03 <TwitterBrasil> Em linha com a campanha do @TSEjusbr para ressaltar a importância da participação de cada um nas #Eleições2018, o Twitter também tem um emoji especial para a hashtag #OVotoImporta pic.twitter.com/yIzwJSNhrU
1030082877235646466 2018-08-16 10:24:30 -03 <TwitterBrasil> A partir de hoje, suas conversas sobre as #Eleições2018 ganharão um emoji especial no Twitter. Basta usar a hashtag para incluir a imagem nos seus Tweets <img alt="Twitter bird icon" data-bbox="695 390 715 405"/> pic.twitter.com/h1djbCuQkr
1029837094959702020 2018-08-15 18:07:51 -03 <TwitterBrasil> Acompanhe hoje, a partir das 20h45, a transmissão ao vivo do #PMBnoCanalBrasil <img alt="Twitter bird icon" data-bbox="715 425 735 440"/> https://twitter.com/canalbrasil/status/1029832787430465544 ...
1029420134564331520 2018-08-14 14:31:00 -03 <TwitterBrasil> Daremos continuidade a esse trabalho nas próximas semanas na medida em que identificarmos outras contas que tentarem Tweetar mesmo estando suspensas. Se você acredita que
```

Figura. 2.4. Chamada e retorno do comando `$ python3 Twint.py -u twitterbrasil`

2.2.3.5. Obter os resultados de uma pesquisa

Via API

O método `search/tweets`¹⁴ retorna os *tweets* que correspondem aos parâmetros da consulta. Estes parâmetros podem incluir palavras-chave, *hashtags*, frases, regiões, nomes de usuários ou *ids*.

A Listagem 2.8 mostra o código Python para obter *tweets* contendo a palavra chave enviada pelo parâmetro *busca*.

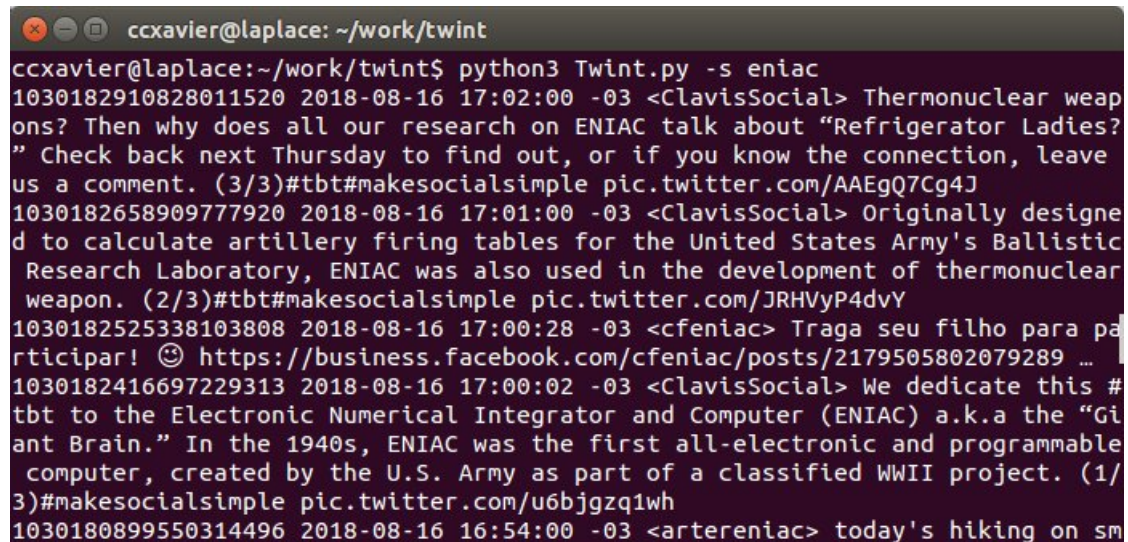
Listagem 2.8. Código em Python para chamada API `search/tweets`.

```
def busca_tweets(busca):
    GET_USR_URL = "https://api.twitter.com/1.1/search/tweets
        .json?q="+busca
    req = oauth_req(GET_USR_URL, TOKEN_ACESSO, CHAVE_ACESSO)
    return req
```

Via Twint

¹⁴<https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html>

O comando a seguir retorna todos os *tweets* contendo a palavra-chave informada como parâmetro. `python3 Twint.py -s [palavra-chave]` . A Figura 2.5 mostra o comando e o retorno do Twint para a busca da palavra-chave "eniac".



```
ccxavier@laplace: ~/work/twint
ccxavier@laplace:~/work/twint$ python3 Twint.py -s eniac
1030182910828011520 2018-08-16 17:02:00 -03 <ClavisSocial> Thermonuclear weapons? Then why does all our research on ENIAC talk about "Refrigerator Ladies?" Check back next Thursday to find out, or if you know the connection, leave us a comment. (3/3)#tbt#makesocialsimple pic.twitter.com/AAEgQ7Cg4J
1030182658909777920 2018-08-16 17:01:00 -03 <ClavisSocial> Originally designed to calculate artillery firing tables for the United States Army's Ballistic Research Laboratory, ENIAC was also used in the development of thermonuclear weapon. (2/3)#tbt#makesocialsimple pic.twitter.com/JRHVYP4dvY
1030182525338103808 2018-08-16 17:00:28 -03 <cfeniac> Traga seu filho para participar! ☺ https://business.facebook.com/cfeniac/posts/2179505802079289 ...
1030182416697229313 2018-08-16 17:00:02 -03 <ClavisSocial> We dedicate this #tbt to the Electronic Numerical Integrator and Computer (ENIAC) a.k.a the "Giant Brain." In the 1940s, ENIAC was the first all-electronic and programmable computer, created by the U.S. Army as part of a classified WWII project. (1/3)#makesocialsimple pic.twitter.com/u6bjgq1wh
1030180899550314496 2018-08-16 16:54:00 -03 <artereniac> today's hiking on sm
```

Figura. 2.5. Chamada e retorno do comando `$ python3 Twint.py -s eniac`

2.3. Análise de Polaridade

De acordo com Devika *et al.* [Devika et al. 2016] a Análise de Polaridade (incluindo a Análise de Sentimentos) é um tipo de classificação de texto que se baseia na orientação do sentimento expresso na opinião que ele denota. Em nossa opinião, esta tarefa é uma das aplicações mais interessantes de Análise de Texto.

A principal tarefa da Análise de Sentimento no nível da sentença é encontrar sua polaridade. Por exemplo, um sentimento positivo é atribuído quando o *Tweet* analisado indica felicidade, excitação ou simpatia. Um sentimento negativo está relacionado a raiva, tristeza, situações tristes ou difíceis. Quando nenhuma emoção é sugerida, o texto pode ser classificado como neutro [Lalrempuii and Mittal 2016].

Por causa da linguagem informal e específica, a análise da polaridade das mensagens do Twitter é uma tarefa singular. A classificação das postagens de acordo com o sentimento expresso tem várias aplicações na ciência política, ciências sociais, pesquisa de mercado, etc [Martínez-Cámara et al. 2014, Mejova et al. 2015, Nakov et al. 2016]. Pak e Paroubek [Pak and Paroubek 2010] apresentam razões para o uso do Twitter como um corpus para análise de sentimentos e mineração de opinião, apresentando as seguintes razões para isso:

- As plataformas de *microblogging* são utilizadas por muitas pessoas para expressar seu ponto de vista sobre diferentes tópicos, sendo assim, uma fonte valiosa de opiniões.
- O Twitter contém uma quantidade imensa de postagens de texto que cresce a cada dia. Desta forma, o corpus coletado pode ser bastante grande.

Table 2.1. Exemplo de classificação de polaridade de *tweets*

Usuário	@ivetesangalo
<i>Tweet</i>	Estou muito feliz e muito agradecida por todo esse amor ♥♥♥
Polaridade	Positiva

Usuário	@EPTC_POA
<i>Tweet</i>	Neste momento, bem complicado o acesso a Rodoviária no Largo Vespasiano Julio Veppo, pelo Túnel da Conceição.
Polaridade	Negativa

- A audiência do Twitter varia de usuários comuns à celebridades, representantes de empresas, políticos e até mesmo presidentes de países. Portanto, é possível coletar mensagens de texto de usuários de diferentes grupos sociais e de interesses.
- A audiência do Twitter é representada por usuários de vários países. É possível coletar dados em diferentes idiomas.

Para exemplificar como funciona a classificação de polaridade de *tweets*, a Tabela 2.1 apresenta exemplos de classificação positiva e negativa.

2.3.1. Abordagens

Existem diferentes abordagens para a realização desta tarefa. As principais são:

2.3.1.1. Abordagem baseada em regras

Nesta abordagem são definidas regras ou padrões para compreender as opiniões sobre o texto. Funcionamento [Devika et al. 2016]:

- Para cada sentença:
 - Faz a tokenização.
 - Inicia com pontuação neutra (0).
 - Para cada padrão encontrado, aplica uma classificação (pontuação).
- A sentença é considerada positiva se a pontuação de polaridade final for maior que zero ou negativa se a pontuação geral for menor que zero.

2.3.1.2. Abordagem baseada em léxico

Técnicas baseadas em léxico partem do pressuposto de que a polaridade ou o sentimento expresso por uma frase ou documento pode ser identificada pelas polaridades

das unidades lexicais que a compõem. Segundo Gómez Molina [Gómez Molina 2004], a unidade lexical (item léxico) é a unidade de significado no léxico mental, que serve como um veículo para a cultura do idioma e pode ser composta de uma ou mais palavras (cabeça, guarda-chuva, dinheiro sujo, etc.).

Para realizar a análise de sentimentos via métodos léxicos, é necessário, inicialmente, efetuar o pré-processamento dos textos utilizados. Este pré-processamento visa reduzir o volume dos dados, antes de iniciar a execução das etapas de análise. Após o pré-processamento, o método de análise de sentimentos precisa representar cada texto e suas palavras. Nessas representações, cada palavra é representada por meio de um peso. Esse peso pode ser simplesmente sua frequência, ou, por exemplo, o valor TF-IDF. Em vista da métrica utilizada para definição do peso das palavras analisadas, o peso da palavra positiva ou negativa, aumenta proporcionalmente à medida que aumenta o número de ocorrências dela no documento. Esse valor pode ser equilibrado pelo inverso da frequência definida para cada termo. Com isto, é possível distinguir o fato de alguns termos serem geralmente mais comuns que outros no âmbito positivo ou negativo [de Souza et al. 2017].

2.3.1.3. Abordagem baseada em aprendizado de máquina

As estratégias utilizadas nesta abordagem funcionam através da aplicação de um algoritmo de treino. Este algoritmo primeiro treina a si mesmo utilizando um conjunto de dados de treinamento e somente depois realiza o aprendizado em si. Sendo assim, as técnicas de aprendizado de máquina primeiro treinam o algoritmo com algumas entradas específicas, para depois trabalhar com novos dados desconhecidos [Devika et al. 2016].

Nós iremos focar nesta abordagem para realizar a classificação da polaridade de *tweets*. Desta forma, avaliaremos os seguintes classificadores:

- Máxima Entropia: de acordo com [Ratnaparkhi 1997] “segundo o princípio da máxima entropia, a distribuição correta é aquela que maximiza a entropia ou sujeito incerto às restrições que representam a “evidência”, isto é, os fatos conhecidos pelo experimentador”. Este princípio é frequentemente invocado na criação de modelos, assumindo que os dados observados em si são a informação testável.
- Naive Bayes: é um modelo de probabilidade que assume a independência entre os recursos de entrada, baseado na aplicação do teorema de Bayes. Ele assume que a presença de uma *feature* específica não se relaciona com a existência de qualquer outra *feature* [John and Langley 1995].
- *Support Vector Machine*¹⁵ (SVM): um algoritmo de classificação supervisionado que foi usado extensivamente e com sucesso para a tarefa de classificação de texto [Pawar and Gawande 2012]. Dado um conjunto de exemplos de treino, cada um marcado como pertencente a uma categoria, o algoritmo de treinamento SVM cria um modelo que atribui novos exemplos a uma categoria. Um modelo de SVM é uma representação dos exemplos como pontos em um hiperplano. O que o algoritmo faz

¹⁵Máquina de vetores de suporte

é encontrar uma linha de separação (hiperplano) entre dados de duas classes (Representado na Figura 2.6. Essa linha busca maximizar a distância entre os pontos mais próximos em relação a cada uma das classes [D'Andrea et al. 2015].

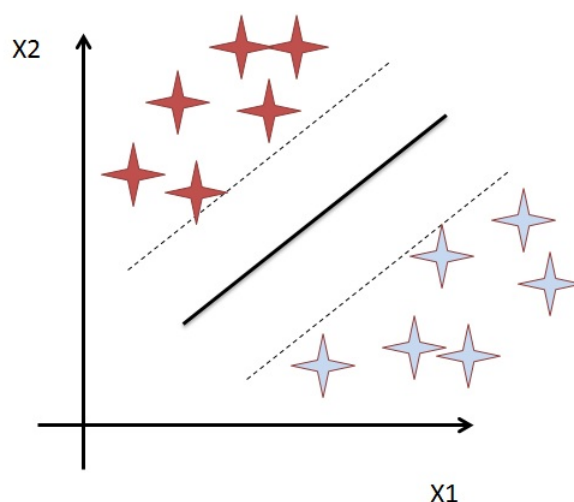


Figura. 2.6. Representação gráfica de um hiperplano (linha central).

2.3.2. Exemplo Prático

Buscando aplicar os conceitos vistos anteriormente na prática, vamos analisar e editar um classificador de polaridade implementado em Python usando a biblioteca NLTK ¹⁶.

A biblioteca Python NLTK implementa ferramentas para processamento de linguagem natural. Ela fornece interfaces para corpora e recursos lexicais como a WordNet, juntamente com um conjunto de bibliotecas de processamento de texto para classificação, tokenização, *stemming*, *tagging*, *parsing* e análise semântica [Bird et al. 2009].

Classificação é a tarefa de aplicar o rótulo correto para uma determinada entrada. Em tarefas de classificação básica, cada entrada é considerada isoladamente de todas as outras entradas e o conjunto de rótulos é definido antecipadamente. Neste exemplo iremos considerar a análise de polaridade como uma tarefa de classificação onde os *tweets* podem ser classificados como Positivos, Negativos ou Neutros. Mais especificamente, trabalharemos com "classificação supervisionada". Um classificador é denominado supervisionado quando utiliza um corpora de treinamento. A Figura 2.7 representa o seu *framework* que é dividido em duas etapas definidas: treino e aprendizado, conforme detalhamento abaixo.

- A Treinamento: um extrator de *features* converte cada valor de entrada em um *feature set*. Pares de *feature sets* e rótulos alimentam o algoritmo de aprendizado de máquina para gerar um modelo.
- B Previsão: o mesmo extrator de *features* é usado para converter novas entradas (não presentes nos *feature sets*). Os *feature sets* são alimentados no modelo, o que gera os novos rótulos (classificação).

¹⁶<http://www.nltk.org/>

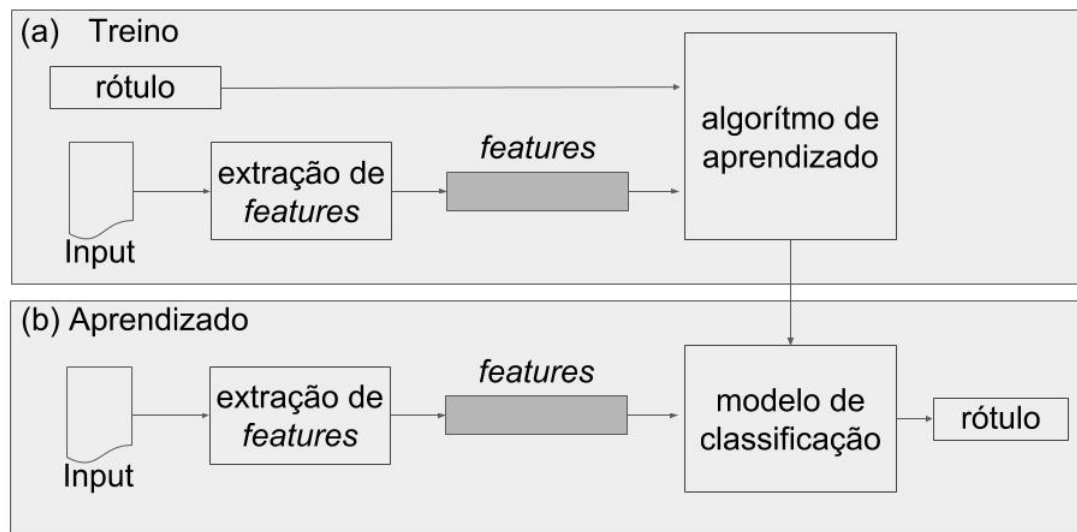


Figura. 2.7. Framework de um classificador supervisionado - baseado em [Bird et al. 2009].

Os classificadores de aprendizado de máquina geralmente exigem que a entrada de texto seja representada como um vetor de comprimento fixo. Provavelmente o mais comum destes vetores seja o *bag-of-words* devido à sua simplicidade, eficiência e, muitas vezes, precisão [Le and Mikolov 2014]. Neste modelo, o texto é representado como um saco (*bag*) ou conjunto de suas palavras, desconsiderando a gramática e até a ordem das palavras, mas mantendo a multiplicidade. O exemplo de um vetor *bag-of-words* para a frase Ana gosta de Zeca, Zeca gosta de Lia e Lia não gosta de ninguém seria ["Ana":1, "gosta":3, "de":3, "Zeca":2, "Lia":2, "e":1, "não":1, "ninguém":1]

Nossos experimentos usam este modelo como *feature set*.

2.3.2.1. Extração de Features

Na listagem 2.9 podemos ver o código Python que implementa a extração das *features*, ou seja, criação do vetor de entrada no modelo *bag-of-words* que será utilizado tanto pelo algoritmo de aprendizado, como pelo modelo de classificação, conforme ilustrado na Figura 2.7

A função `divide()` recebe como entrada um vetor de frases e retorna um vetor de palavras para cada frase no vetor de entrada. Por exemplo, considerando um vetor de entrada ["Trânsito acentuado nos dois sentidos da Av. Carlos Gomes x Campos Sales."] a função retorna o vetor ['trânsito', 'acentuado', 'nos', 'dois', 'sentidos', 'da', 'av.', 'carlos', 'gomes', 'x', 'campos', 'sales.']

A função descrita na Listagem 2.9 cria o vetor *bag-of-words*

Listagem 2.9. Código em Python implementando extração de features

```
def divide(dados):
    dados_new = []
    for palavra in dados:
        palavra_filter = [i.lower() for i in palavra.split()]
        dados_new.append(palavra_filter)
    return dados_new[0]

def bag_of_words(palavras):
    return dict([(palavra, palavras.count(palavra)) for
                 palavra in palavras])
```

2.3.2.2. Treino

Na listagem 2.10 podemos ver o código Python que implementa o treinamento de três tipos de classificadores supervisionados utilizando a biblioteca NLTK: SVM, Naive Bayes e Máxima Entropia.

Os dados de treino são *tweets* da conta @EPTC_POA¹⁷ rotulados manualmente como 'pos' (positivo), 'neg' (negativo) e 'neu' (neutro). Estes dados estão disponíveis para *download* em <https://github.com/clarissacastella/twittercourse/>.

A função utiliza as *features bag-of-words* geradas pelas funções da listagem 2.9 e retorna um modelo treinado para cada um dos três algoritmos de classificação.

Listagem 2.10. Código em Python implementando treinamento dos classificadores.

```
def treina_classificadores():
    posdados = []
    with open('./dadostreino/train_EPTC_POA_v3nbal_1.data',
              'rb') as myfile:
        reader = csv.reader(myfile, delimiter=',')
        for val in reader:
            posdados.append(val[0])
    negdados = []
    with open('./dadostreino/train_EPTC_POA_v3nbal_0.data',
              'rb') as myfile:
        reader = csv.reader(myfile, delimiter=',')
        for val in reader:
            negdados.append(val[0])
    neudados = []
    with open('./dadostreino/train_EPTC_POA_v3nbal_2.data',
              'rb') as myfile:
        reader = csv.reader(myfile, delimiter=',')
        for val in reader:
```

¹⁷https://twitter.com/EPTC_POA

```

        neudados.append(val[0])
    negfeats = [(bag_of_words(f), 'neg') for f in divide(
        negdados)]
    posfeats = [(bag_of_words(f), 'pos') for f in divide(
        posdados)]
    neufeats = [(bag_of_words(f), 'neu') for f in divide(
        neudados)]
    treino = negfeats + posfeats + neufeats
    # 'Maximum Entropy'
    classificadorME = MaxentClassifier.train(treino, 'GIS',
        trace=0, encoding=None, labels=None,
        gaussian_prior_sigma=0, max_iter = 1)
    # SVM
    classificadorSVM = SklearnClassifier(LinearSVC(), sparse
        =False)
    classificadorSVM.train(treino)
    # Naive Bayes
    classificadorNB = NaiveBayesClassifier.train(treino)
    return ([classificadorME, classificadorSVM,
        classificadorNB])

```

2.3.2.3. Classificação

Os três modelos criados pela função `treina_classificadores` são utilizados no código da Listagem 2.11 que apresenta a função que classifica uma sentença de entrada (no caso um *Tweet*), atuando conforme os passos apresentados na figura 2.7.

Listagem 2.11. Código em Python implementando um classificador de polaridade.

```

def classifica(sentencas, classificadores):
    ret = []
    for s in sentencas:
        c = divide([s])
        feats = bag_of_words(c[0])
        classificacao = []
        classificacao.append(classificadores[1].classify(
            feats))
        classificacao.append(classificadores[2].classify(
            feats))
        classificacao.append(classificadores[0].classify(
            feats))
        ret.append(classificacao)
    return ret

```

Por exemplo, quando a função `classifica` recebe como entrada um vetor `['Fluxo muito congestionado na Osvaldo Aranha no acesso para`

o Túnel. Agora, tá chovendo também. Então, atenção!’,
'Não use o celular ao volante, 80% da sua atenção é
desviada'] com duas sentenças, ela retorna o vetor de resultados [['neg', 'pos',
'neg'], ['neu', 'neu', 'neu']]. Desta forma, o primeiro *Tweet* foi classi-
ficado com polaridade negativa pelo classificador Máxima Entropia, positiva pelo SVM e
negativa pelo Naive Bayes. Ou seja, os classificadores Máxima Entropia e Naive Bayes
classificaram corretamente a sentença e o classificador SVM não. Já o segundo *Tweet* foi
classificado corretamente como neutro pelos três classificadores.

Para maiores detalhes, a implementação completa do classificador de polaridade
encontra-se em <https://github.com/clarissacastella/twittercourse>,
arquivo `polaridade.py`.

2.4. Extração de Entidades

A Extração de Entidades (EE), também conhecida como Reconhecimento de En-
tidades Nomeadas (REN), é uma tarefa de Extração de Informações que consiste na iden-
tificação de referências feitas a determinadas entidades e sua classificação. Essa tarefa é
frequentemente uma das primeiras etapas na análise semântica de um texto, posto que as
entidades mencionadas transmitem bastante informação sobre o conteúdo do texto em si.

O formato reduzido das mensagens do Twitter impõe algumas dificuldades em
seu processamento. Por esse motivo, os métodos de EE que são aplicados em outros tipos
de texto podem não funcionar muito bem nos *tweets*. Por exemplo, Locke [Locke 2009]
mostra que um anotador de última geração como o *Stanford NER*¹⁸, tem seu desempenho
bastante reduzido quando aplicado a textos do Twitter.

O tamanho das mensagens limitado à 280 caracteres apresenta uma dificuldade
para a tarefa de EE. Textos curtos oferecem pouca informação contextual para a identi-
ficação de entidades. Além disso, recursos importantes para identificar nomes próprios,
como capitalização, não são utilizados com rigor nos textos do Twitter. É muito comum a
sub-capitalização das palavras, ou seja, nenhuma palavra capitalizada no texto, bem como
a supercapitalização, onde várias ou todas as palavras estão capitalizadas, geralmente com
a intenção de denotar intensidade ao conteúdo do texto. Assim, geralmente, os métodos
de EE devem ser adaptados a esse contexto.

Para demonstrar como a EE de *tweets* funciona, o exemplo abaixo mostra quais
entidades podem ser extraídas do Tweet da @EPTC_POA listado na Tabela 2.1 “*Neste
momento, bem complicado o acesso a Rodoviária no Largo Vespasiano Julio Veppo, pelo
Túnel da Conceição*” usando o formato [entidade (classificação)].

```
[Rodoviária (LOC)] [Largo Vespasiano Julio Veppo (LOC)] [Tú-  
nel da Conceição (LOC)]
```

Os sistemas de EE mais recentes têm utilizado técnicas de aprendizado de máquina,
de modo que esta se tornou a principal técnica de abordagem, em contraste com os sis-
temas mais antigos que utilizavam regras manualmente codificadas e heurísticas para
efetuar esta tarefa. Desta forma, EE vem sendo tratada como uma tarefa de rotulagem

¹⁸<https://nlp.stanford.edu/software/CRF-NER.shtml>

sequencial [Souza 2012].

As ferramentas de EE a partir do Twitter, em geral, usam técnicas do estado-da-arte associadas a estratégias de otimização, como heurísticas de pré-processamento [Ritter et al. 2011] ou métodos semi-supervisionados de aprendizado [Liu 2010] que exploram a grande quantidade de dados não anotados disponíveis.

2.4.1. Exemplo Prático

Buscando aplicar os conceitos vistos na prática, vamos explorar uma solução de aprendizado de máquina que extrai entidades de implementada em Python. Esta ferramenta utiliza a biblioteca SpaCy¹⁹, uma biblioteca de código aberto para o processamento avançado de linguagem natural. A SpaCy é uma solução baseada em *deep learning* que interage com as bibliotecas do ecossistema de Inteligência Artificial do Python. A biblioteca usa um sistema de EE estatístico, que atribui rótulos a extensões contíguas de *tokens*²⁰.

2.4.1.1. Treino

Os modelos gerados pela biblioteca são estatísticos e cada decisão, como por exemplo se uma palavra é uma entidade, é uma previsão. Essa previsão é baseada nos exemplos que o modelo viu durante o treinamento. Para treinar um modelo é necessário um conjunto de dados de treinamento que consistem de exemplos de texto e de rótulos que o modelo deve prever. Como treinar um modelo não é simplesmente memorizar exemplos, mas gerar um padrão que possa ser generalizado em outros exemplos, os dados de treinamento devem ser representativos dos dados que serão processados.

Na listagem 2.12 podemos ver o código Python que implementa transformando os dados de treino contidos no arquivo `./data/dadosTreinoLoc.txt` para o formato de entrada do modelo. Desta forma, a função `cria_dados_treino` transforma o conteúdo de um arquivo texto contendo uma lista de *tweets* e os locais citados neles em um vetor contendo uma tupla. A Tabela 2.2 apresenta um exemplo do funcionamento da função. A linha Entrada consiste em um trecho do arquivo de entrada e a linha Saída representa a tupla correspondente ao trecho dentro do vetor de saída.

Listagem 2.12. Código em Python formatando os dados de treino.

```
def cria_dados_treino(arq='./data/dadosTreinoLoc.txt') :
    dados_treino = []
    fin = open(arq, 'rb')
    n=0
    post = u''
    for val in fin:
        d = {}
        n = n + 1
```

¹⁹<https://spacy.io/>

²⁰Os *tokens* usualmente correspondem as unidades lexicais de um texto, i.e. aproximadamente às palavras do texto. Por exemplo os tokens da frase “Complicado o trânsito” são [complicado] [o] [trânsito]

```

if (n % 2 == 1) :
    post = val.replace('\n', '')
else :
    d['entidades'] = ast.literal_eval(val.
        replace('\n', ''))
    dados_treino.append((post, d))
fin.close()
return dados_treino

```

Table 2.2. Exemplo do funcionamento da função `cria_dados_treino`.

Entrada	... 17h53 - ATENÇÃO! Acidente entre carro e moto na R. Dom Pedro II, sentido sul/norte, próximo a R. Barão do Cotegipe. [(48,63,'LOC'), (94,114,'LOC')]
Saída	{ [...('17h53 - ATENÇÃO! Acidente entre carro e moto na R. Dom Pedro II, sentido sul/norte, próximo a R. Barão do Cotegipe.', ({'entidades': { [(48,63,'LOC'), (94,114,'LOC')] } }]] }

Na listagem 2.13 podemos ver o código Python que implementa a criação de um novo modelo SpaCy para realizar EE e o treina com os dados importados pela função `cria_dados_treino` (listagem 2.12) seguindo os seguintes passos:

- Cria modelo em branco.
- Adiciona ao modelo o *pipeline* NER, responsável por executar a EE.
- Lê os dados de treino.
- Treina o modelo.

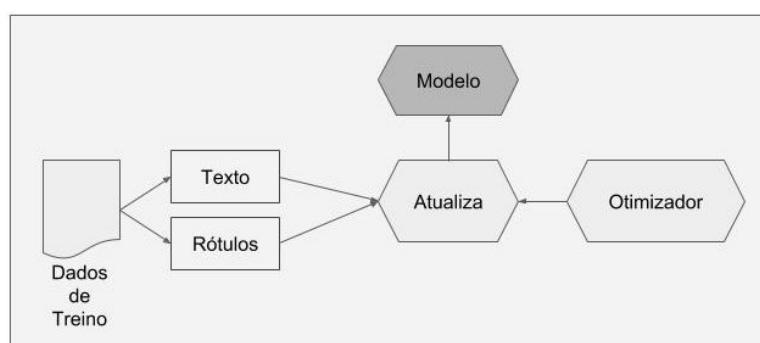


Figura. 2.8. Fluxo de treino do EE - adaptado de [SpaCy 2018]

O treino é feito em várias iterações, como pode ser visto na Imagem 2.8. Em cada iteração os dados de treinamento são embaralhados para que o modelo não faça generalizações com base na ordem dos exemplos. Também é definida uma taxa de desistência (*loss*) para "descartar" recursos e representações individuais aleatoriamente. Por exemplo, um abandono de 0,25 significa que cada recurso ou representação interna tem uma probabilidade de 1/4 de ser descartado.

Listagem 2.13. Código que treina um novo modelo de EE.

```
dir = "./modeloEE" # cria modelo em branco
dir = Path(dir)
nlp = spacy.blank('pt')
dir.mkdir()
ner = nlp.create_pipe('ner')
nlp.add_pipe(ner, last=True)

import dados_treino as tr
dados_treino = tr.cria_dados_treino('./data/testP.txt') #
    dados de treino

for _, annotations in dados_treino:
    for ent in annotations.get('entities'):
        ner.add_label(ent[2])

# treina EE
optimizer = nlp.begin_training()
n_iter = 2 #nro iteracoes
for itn in range(n_iter):
    c = 0
    losses = {}
    random.shuffle(dados_treino) #embaralha para nao viciar
    for text, annotations in dados_treino:
        c = c + 1
        nlp.update(
            [unicode(text)], # batch de textos
            [annotations], # batch de anotacoes
            correspondentes aos textos
            drop=0.25,
            sgd=optimizer,
            losses=losses)
    print(losses)

nlp.to_disk(dir) #salva modelo
```

2.4.1.2. Aprendizado

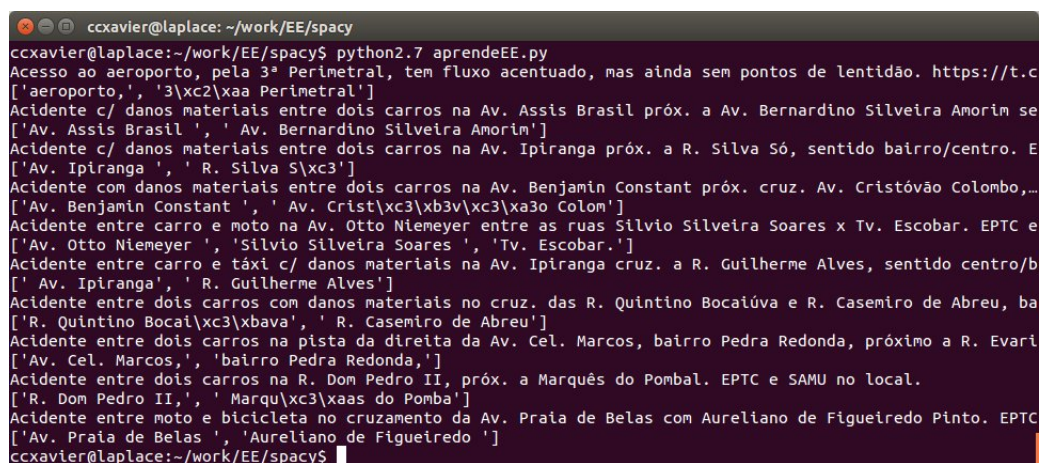
Na listagem 2.14 podemos ver o código Python que implementa a EE a partir do modelo SpaCy gerado anteriormente seguindo os seguintes passos:

- Carrega modelo.
- Lê arquivo com posts.
- Para cada post: aplica o modelo de EE.

Listagem 2.14. Código que implementa o EE usando o modelo criado.

```
modelo_dir = Path ("../modeloEE" )
modelo = spacy.load(modelo_dir) # roda modelo
fin = open('./data/postsP.txt', 'rb')
for text in fin:
    doc_model = modelo(unicode(text))
    locs = []
    for ent in doc_model.ents:
        aux = (ent.start_char, ent.end_char, str(ent.
            label_))
        locs.append(text[ent.start_char:ent.end_char])
    print (text,locs)
```

A Figura 2.9 mostra a chamada e o retorno do programa correspondente ao código listado em 2.14.



The image shows a terminal window with the following content:

```
ccxavier@laplace: ~/work/EE/spacy
ccxavier@laplace:~/work/EE/spacy$ python2.7 aprendeEE.py
Acesso ao aeroporto, pela 3ª Perimetral, tem fluxo acentuado, mas ainda sem pontos de lentidão. https://t.c
[['aeroporto', '3\x2\xaa Perimetral']]
Acidente c/ danos materiais entre dois carros na Av. Assis Brasil próx. a Av. Bernardino Silveira Amorim se
[['Av. Assis Brasil ', ' Av. Bernardino Silveira Amorim']]
Acidente c/ danos materiais entre dois carros na Av. Ipiranga próx. a R. Silva Só, sentido bairro/centro. E
[['Av. Ipiranga ', ' R. Silva S\x3']]
Acidente com danos materiais entre dois carros na Av. Benjamin Constant próx. cruz. Av. Cristóvão Colombo,...
[['Av. Benjamin Constant ', ' Av. Crist\x3v\x3a3o Colom']]
Acidente entre carro e moto na Av. Otto Niemeyer entre as ruas Silvío Silveira Soares x Tv. Escobar. EPTC e
[['Av. Otto Niemeyer ', 'Silvío Silveira Soares ', 'Tv. Escobar.']]
Acidente entre carro e táxi c/ danos materiais na Av. Ipiranga cruz. a R. Guilherme Alves, sentido centro/b
[[' Av. Ipiranga', ' R. Guilherme Alves']]
Acidente entre dois carros com danos materiais no cruz. das R. Quintino Bocaiúva e R. Casemiro de Abreu, ba
[['R. Quintino Bocai\x3vava', ' R. Casemiro de Abreu']]
Acidente entre dois carros na pista da direita da Av. Cel. Marcos, bairro Pedra Redonda, próximo a R. Evarí
[['Av. Cel. Marcos', 'bairro Pedra Redonda,']]
Acidente entre dois carros na R. Dom Pedro II, próx. a Marquês do Pombal. EPTC e SAMU no local.
[['R. Dom Pedro II,', ' Marqu\x3aas do Pomba']]
Acidente entre moto e bicicleta no cruzamento da Av. Praia de Belas com Aureliano de Figueiredo Pinto. EPTC
[['Av. Praia de Belas ', 'Aureliano de Figueiredo ']]
ccxavier@laplace:~/work/EE/spacy$
```

Figura. 2.9. Chamada e retorno do comando `$ python2.7 aprendeEE.py`

Para maiores detalhes, a implementação do Extrator de Entidades encontra-se em <https://github.com/clarissacastella/twittercourse>, arquivos `treinaEE.py` e `aprendeEE.py`.

2.5. Dificuldades no processamento de textos do Twitter

Mensagens do Twitter possuem tamanho limitado, contendo até 280 caracteres. Por tal motivo, podemos caracterizar os *tweets* como uma forma de texto curto, similar a textos de outras mídias como mensagens de texto curta (SMS) ou mensagens de telegrama.

Textos curtos possuem características específicas como uma maneira informal de escrita, utilizando-se de abreviações, gírias, etc. Tais características dificultam o seu processamento ou até mesmo seu entendimento por leitores humanos. Além disso, textos curtos não oferecem informações contextuais importantes que são comumente exploradas por diversas ferramentas de Análise de Texto.

Além disso, por ser um serviço na Internet, funcionando em tempo real ou seja *on the fly*, dados provenientes do Twitter apresentam algumas características próprias, como grande volume, contextos incertos e distribuídos em mensagens diferentes (estrutura de fluxo ou *stream*).

Diferentes estratégias foram experimentadas para lidar com as particularidades de textos do Twitter, como:

- Normalização de texto: utilização técnicas de análise automática para converter o texto ruidoso do Twitter em uma variante mais formal da língua, como corrigir erros de grafia;
- Agrupamento de *tweets*: agrupamento automático de *tweets* tratando de um mesmo assunto, ou que estejam contextualmente ligados, de forma a criar textos capazes de fornecer informação contextual mais relevante às ferramentas de Análise de Texto;
- Séries temporais: como os *tweets* possuem uma estrutura de fluxo, i.e. estão deslocados no tempo, alguns métodos para processamento de *tweets* utilizam modelagens baseadas em séries temporais tentando capturar organicamente o contexto dos *tweets* pela sua informação temporal;
- Ferramentas específicas: alguns métodos e ferramentas parecem precisar de estratégias específicas para textos provenientes de *tweets*, que levam em consideração a pobreza contextual e as variações lexicais próprias desses textos.

Cada uma de tais estratégias possui suas vantagens e desvantagens e não existe uma solução estabelecida para tal problema. Por exemplo, a normalização textual permite a utilização de técnicas e ferramentas já bem desenvolvidas que são especializadas no processamento de texto mais formal, como texto jornalístico, entretanto métodos de normalização são ainda bastante imprecisos e não superam as dificuldades de falta de contexto de *tweets*.

2.6. Conclusão

O Twitter é uma fonte valiosa de informações sobre o que as pessoas pensam e sentem sobre os mais variados assuntos. Por esse motivo academia, empresas e organizações de mídia estão procurando maneiras de extrair conhecimento desta ferramenta.

Este minicurso analisou diferentes abordagens para coletar informações dos *tweets*. Primeiro vimos como coletar informações do Twitter utilizando a API pública, bem como a biblioteca Python Twint. Para realizar a extração e classificação semântica desses dados, usamos técnicas de Processamento da Linguagem Natural e Aprendizado de Máquina. Focamos nosso estudo em duas tarefas de análise de texto: Análise de Polaridade e Extração de Entidades.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

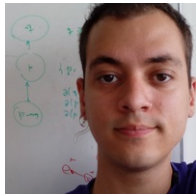
- [Bird et al. 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc."
- [da Cunha Recuero 2003] da Cunha Recuero, R. (2003). Weblogs, webrings e comunidades virtuais. *Revista 404notfound-Revista Eletrônica do Grupo Ciberpesquisa*, 31.
- [Dale 2008] Dale, R. (2008). Where is text analytics today. Presentation to Sydney Data Miners Meeting.
- [D'Andrea et al. 2015] D'Andrea, E., Ducange, P., Lazzerini, B., and Marcelloni, F. (2015). Real-time detection of traffic from twitter stream analysis. *IEEE transactions on intelligent transportation systems*, 16(4):2269–2283.
- [de Souza et al. 2017] de Souza, K. F., Pereira, M. H. R., and Dalip, D. H. (2017). Unilex: Método léxico para análise de sentimentos textuais sobre conteúdo de tweets em português brasileiro. *Abakós*, 5(2):79–96.
- [Devika et al. 2016] Devika, M., Sunitha, C., and Ganesh, A. (2016). Sentiment analysis: A comparative study on different approaches. *Procedia Computer Science*, 87:44–49.
- [Gómez Molina 2004] Gómez Molina, J. R. (2004). La subcompetencia léxico-semántica. *Vademécum para la formación de profesores. Enseñar español como segunda lengua (L2)/lengua extranjera (LE)*, Madrid, SGEL, pages 491–510.
- [John and Langley 1995] John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc.
- [Kumar et al. 2013] Kumar, S., Morstatter, F., and Liu, H. (2013). *Twitter Data Analytics*. Springer Publishing Company, Incorporated.
- [Lalrempuii and Mittal 2016] Lalrempuii, C. and Mittal, N. (2016). Sentiment classification of crisis related tweets using segmentation. In *Proceedings of the International Conference on Informatics and Analytics*, page 89. ACM.
- [Le and Mikolov 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.
- [Liu 2010] Liu, B. (2010). Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2:627–666.
- [Locke 2009] Locke, B. W. (2009). Named entity recognition: Adapting to microblogging. Technical report, University of Colorado.
- [Martínez-Cámara et al. 2014] Martínez-Cámara, E., Martín-Valdivia, M. T., Urena-López, L. A., and Montejo-Ráez, A. R. (2014). Sentiment analysis in twitter. *Natural Language Engineering*, 20(1):1–28.

- [Mejova et al. 2015] Mejova, Y., Weber, I., and Macy, M. W. (2015). *Twitter: a digital socioscope*. Cambridge University Press.
- [Nakov et al. 2016] Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F., and Stoyanov, V. (2016). Semeval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1–18.
- [Newman et al. 2016] Newman, N., Fletcher, R., Levy, D. A. L., and Nielsen, R. K. (2016). Reuters institute digital news report 2016. Technical report, Reuters Institute for the Study of Journalism, University of Oxford.
- [Pak and Paroubek 2010] Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326.
- [Pawar and Gawande 2012] Pawar, P. Y. and Gawande, S. (2012). A comparative study on different types of approaches to text categorization. *International Journal of Machine Learning and Computing*, 2(4):423.
- [Ratnaparkhi 1997] Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. *IRCS Technical Reports Series*, page 81.
- [Ritter et al. 2011] Ritter, A., Clark, S., Etzioni, O., et al. (2011). Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1524–1534. Association for Computational Linguistics.
- [Souza 2012] Souza, M. V. d. S. (2012). Mineração de opiniões aplicada a mídias sociais. Master’s thesis, Pontifícia Universidade Católica do Rio Grande do Sul.
- [SpaCy 2018] SpaCy (2018). Training spacy’s statistical models. <https://spacy.io/usage/training>. Accessed: 2018-09-10.
- [Twitter 2018] Twitter (2018). Sobre as apis do twitter. <https://help.twitter.com/pt/rules-and-policies/twitter-api>. Accessed: 2018-09-10.

Biografia resumida dos autores



Clarissa Castellã Xavier é pesquisadora de pós-doutorado na Universidade Federal do Rio Grande do Sul (UFRGS), mestre e doutora em Ciências da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). Começou a pesquisar em Processamento da Linguagem Natural (PLN) em 1999 no Grupo de Pesquisa em PLN da PUCRS. Desde então, trabalhou para empresas multi-culturais em todo o mundo, desenvolvendo ferramentas de processamento de linguagem com foco em mídias sociais e redes. Seu trabalho atual tem como foco a extração semântica de dados na área do transporte urbano a partir de redes sociais. Também é pesquisadora convidada do grupo FORMAS da Universidade Federal da Bahia (UFBA).



Marlo Souza é professor adjunto na Universidade Federal da Bahia (UFBA), mestre em Ciências da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) e doutor em Ciências da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS). Suas atividades de ensino e pesquisa concentram-se nas áreas de computação teórica, representação do conhecimento, lógica aplicada e PLN. Iniciou suas pesquisas em PLN no ano de 2007 no contexto do projeto CoGROO - Corretor Gramatical livre para o OpenOffice, passando posteriormente pelo Grupo de Pesquisa em PLN da Pontifícia Universidade Católica do Rio Grande do Sul, em que estudou métodos de identificação de entidades e mineração de opiniões em textos do Twitter para monitoramento de marcas. Na UFBA, integra o grupo FORMAS trabalhando com métodos de extração de informações semânticas em texto.