06

Classificando os textos e ganhando produtividade na empresa

Classificando os textos e ganhando produtividade na empresa

Agora precisamos implementar o algoritmo que é capaz de ler os nossos textos, criar um dicionário baseado nas palavras que surgiram durante a leitura dos textos, e então, traduzir todos esses textos em arrays sempre do mesmo tamanho que indica quantas vezes cada palavra aparece. Mas por que precisamos verificar quantas vezes uma palavra aparece em um determinado texto? É justamente pelo fato de concluírmos que se uma palavra como por exemplo, a palavra "amor", aparece constantemente em um texto, provavelmente esse texto é relacionado a um assunto pessoal, sobre quem a pessoa ama, seus filhos, seus pais ou qualquer outro assunto pessoal relacionado à palavra "amor". Porém, dentre esses textos, se tiver algum assunto sobre "compre uma retroescavadeira", aparentemente é um *spam*, pois eu não utilizo uma retroescavadeira, portanto, não faz sentido algum esse e-mail pra mim. Então perceba que a frequência que as palavras aparecerem, ou seja, a quantidade de vezes, pode nos indicar sobre o que o texto está falando, em outras palavras, é relacionado a um problema técnico ou comercial ou cobrança ou certificação... Observe que ao invés de ficarmos criando a árvore de opções, isto é, nós mesmo, ficarmos definindo quais são as palavras chaves, deixamos o próprio algoritmo realizar essa tarefa.

Vamos então implementar esse algoritmo, começaremos criando um arquivo python chamado `classificando_emails.py`, salve-o dentro da pasta onde estão os nossos arquivos python. Primeiramente, definiremos os textos que utilizaremos durante a classificação:

```
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"
```

Vamos rodar rodar o nosso arquivo. Nesse exato momento, você pode estar se perguntando sobre qual é o sentido de rodar um arquivo python que não tem nenhum tipo de lógica ou algoritmo, é justamente para sempre verificarmos logo de cara se existe algum problema no código que escrevemos. Vejamos se tudo ocorre como o esperado executando esse arquivo com apenas essas variáveis:

```
> python classificando_emails.py
  File "classificando_emails.py", line 2
SyntaxError: Non-ASCII character '\xc3' in file classificando_emails.py on line 2, but no encoding c
```

Observe que ocorreu um erro na linha 2, que é justamente onde está a variável `texto2`. O erro nos informa que foi um erro de *syntax* indicando que existe um caracter que não é ASCII. Vejamos o que aconteceu exatamente:

```
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
```

Note que nessa frase temos um acento na palavra "exercício", ou seja, um acento não é um caracter ASCII. No python, se queremos utilizar o padrão UTF-8 que abragem um conjunto de caracteres ao qual utilizamos no dia-a-dia, precisamos informar ao python. Para isso, basta adicionarmos o código `#!-*- coding: utf8 -*-` no nosso arquivo:

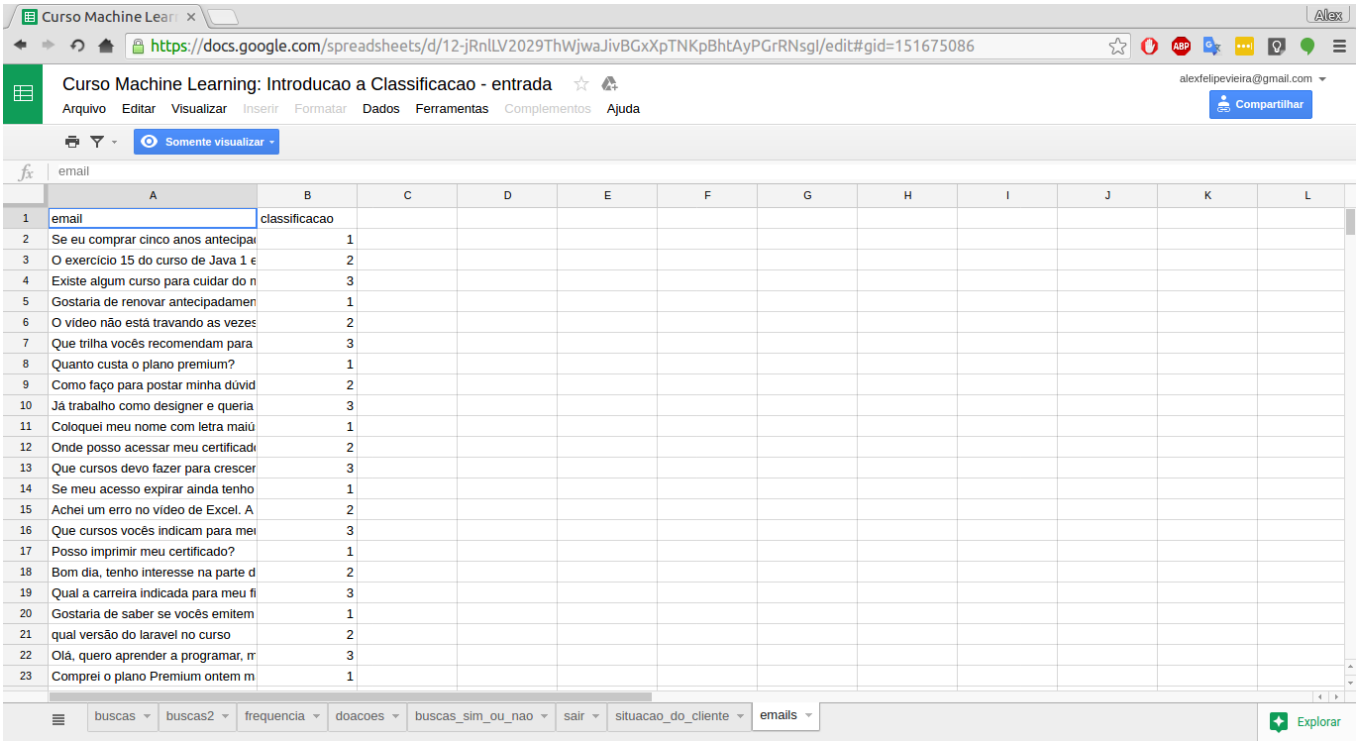
```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
```

```
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"
```

Vamos tentar rodar novamente o nosso arquivo python:

```
> python classificando_emails.py
  File "classificando_emails.py", line 2
SyntaxError: Non-ASCII character '\xc3' in file classificando_emails.py on line 2, but no encoding d
> python classificando_emails.py
>
```

Veja que agora ele funciona normalmente, pois a partir do momento que adicionamos o trecho indicando que queremos que ele utilize o UTF-8, o python já entende que tudo abaixo será interpretado com o enconding UTF-8. Entretanto, existe um pequeno detalhe, que é a quantidade de variáveis que estamos criando para os nossos textos, por exemplo, atualmente temos 3 textos, e se forem 10? Ou então, 100? Escreveremos tudo dentro desse arquivo? Não faz muito sentido, portanto, iremos extrair esses textos de um arquivo CSV. Esses dados estarão nas [planilhas do Google Spreadsheets (http://bit.ly/22lUw1z)](http://bit.ly/22lUw1z) na aba **emails**:



Salve esses dados como `emails.csv` dentro da pasta onde estão os arquivos python. Vejamos o conteúdo do nosso arquivo CSV:

```
email,classificacao
"Se eu comprar cinco anos antecipados, eu ganho algum desconto?",1
O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?,2
Existe algum curso para cuidar do marketing da minha empresa?,3
"Gostaria de renovar antecipadamente meu plano, como posso fazer?",1
...
gosta de design mas não é fã de programação. Ele deve fazer lógica?",3
```

Note que nesse arquivo temos o texto que refere-se ao e-mail e um número que corresponde à sua categoria. Precisamos fazer a leitura desse arquivo. Como fazíamos antes? Utilizamos o pandas, portanto, utilizaremos o pandas também:

```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
```

Vamos imprimir a variável `classificacoes` e verificar o seu valor:

```
print classificacoes
```

Testando o nosso arquivo `classificando_emails.py` :

```
> python classificando_emails.py
                                                email  classificacao
0   Se eu comprar cinco anos antecipados, eu ganho...              1
1   O exercício 15 do curso de Java 1 está com a r...              2
2   Existe algum curso para cuidar do marketing da...              3
3   Gostaria de renovar antecipadamente meu plano,...              1
4   O vídeo não está travando as vezes no meu nave...              2
5   Que trilha vocês recomendam para quem quer com...              3
6                       Quanto custa o plano premium?              1
7        Como faço para postar minha dúvida no fórum?              2
8   Já trabalho como designer e queria aprender ma...              3
9   Coloquei meu nome com letra maiúscula no certi...              1
10                Onde posso acessar meu certificado?              2
11  Que cursos devo fazer para crescer na minha ca...              3
12  Se meu acesso expirar ainda tenho acesso aos m...              1
13  Achei um erro no vídeo de Excel. A fórmula usa...              2
14  Que cursos vocês indicam para meu filho de 14 ...              3
15                   Posso imprimir meu certificado?              1
16  Bom dia, tenho interesse na parte de IOS, pore...              2
17           Qual a carreira indicada para meu filho?              3
18  Gostaria de saber se vocês emitem certificado ...              1
19                   qual versão do laravel no curso              2
20  Olá, quero aprender a programar, mas não sei p...              3
21  Comprei o plano Premium ontem mas Queria mudar...              1
22    Como faço para me tornar um moderador no Alura?              2
23  estou interessado em python minha duvida é se ...              3
24  Usei o cartão do meu pai e ele cancelou o paga...              1
25  Como faço para ser convidado a me tornar um mo...              2
26  Me cadastrei com o email antigo do paypal, mas...              1
27  Quais os passos para me tornar um desenvolvedo...              3
28  Tenho o plano premiun de renovar o plano Premi...              1
29  Quero ajudar mais a comunidade do Alura e me t...              2
30  Terminei a faculdade em sistemas de informação...              3
31               Além do paypal posso pagar com moip?              1
32  Encontrei um erro de digitação em uma explicaç...              2
33  Posso efetuar o pagamento a vista via transfer...              1
```

```
34  Boa noite, equipe do Alura! Senti a necessidad...              3
35  Usei a conta do paypal de meus pais para me ca...              1
36  Desde já, desejo-te um ótimo início de semana,...              2
37  Gostaria de saber qual o melhor curso para eu ...              3
38  Posso usar o cartão de meu pai para efetuar o ...              1
39  Trabalhando com o framework Django não ficamos...              2
40  É necessário aprender html, css e javascrip em...              3
41  Estou com dificuldade para fazer o pagamento, ...              1
42  Gostaria de saber qual o melhor caminho para m...              3
```

Repara que ele exibe os e-mails e a classificação e 42 e-mails no total. Qual é o próximo passo? Precisamos pegar a coluna `email` que são os dados que iremos trabalhar, como fazemos mesmo no pandas? Pedimos a coluna:

```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
```

Vamos verificar se os e-mails vieram corretamente:

```
print textosPuros
```

Executando o nosso arquivo novamente:

```
> python classificando_emails.py
0     Se eu comprar cinco anos antecipados, eu ganho...
1     O exercício 15 do curso de Java 1 está com a r...
2     Existe algum curso para cuidar do marketing da...
3     Gostaria de renovar antecipadamente meu plano,...
4     O vídeo não está travando as vezes no meu nave...
5     Que trilha vocês recomendam para quem quer com...
6                          Quanto custa o plano premium?
7          Como faço para postar minha dúvida no fórum?
8     Já trabalho como designer e queria aprender ma...
9     Coloquei meu nome com letra maiúscula no certi...
10                   Onde posso acessar meu certificado?
11    Que cursos devo fazer para crescer na minha ca...
12    Se meu acesso expirar ainda tenho acesso aos m...
13    Achei um erro no vídeo de Excel. A fórmula usa...
14    Que cursos vocês indicam para meu filho de 14 ...
15                       Posso imprimir meu certificado?
16    Bom dia, tenho interesse na parte de IOS, pore...
17              Qual a carreira indicada para meu filho?
18    Gostaria de saber se vocês emitem certificado ...
19                       qual versão do laravel no curso
20    Olá, quero aprender a programar, mas não sei p...
21    Comprei o plano Premium ontem mas Queria mudar...
22      Como faço para me tornar um moderador no Alura?
23    estou interessado em python minha duvida é se ...
```

```
24    Usei o cartão do meu pai e ele cancelou o paga...
25    Como faço para ser convidado a me tornar um mo...
26    Me cadastrei com o email antigo do paypal, mas...
27    Quais os passos para me tornar um desenvolvedo...
28    Tenho o plano premiun de renovar o plano Premi...
29    Quero ajudar mais a comunidade do Alura e me t...
30    Terminei a faculdade em sistemas de informação...
31                 Além do paypal posso pagar com moip?
32    Encontrei um erro de digitação em uma explicaç...
33    Posso efetuar o pagamento a vista via transfer...
34    Boa noite, equipe do Alura! Senti a necessidad...
35    Usei a conta do paypal de meus pais para me ca...
36    Desde já, desejo-te um ótimo início de semana,...
37    Gostaria de saber qual o melhor curso para eu ...
38    Posso usar o cartão de meu pai para efetuar o ...
39    Trabalhando com o framework Django não ficamos...
40    É necessário aprender html, css e javascrip em...
41    Estou com dificuldade para fazer o pagamento, ...
42    Gostaria de saber qual o melhor caminho para m...
Name: email, dtype: object
```

Conseguimos pegar os textos, porém, precisamos das palavras separadamente desses textos. A abordagem mais simples seria quebrar esses textos em espaços. Como fazer isso? Podemos pedir para o pandas tratar cada texto como uma *string*, e então, separar essas strings por espaço em branco:

```
# restante do código

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.split(' ')
print textosQuebrados
```

Observe que a variável `textosQuebrados` refere-se aos textos que contém as palavras separadas por espaço. Vamos verificar novamente como ficou o resultado:

```
> python classificando_emails.py
0     [Se, eu, comprar, cinco, anos, antecipados,, e...
1     [O, exercício, 15, do, curso, de, Java, 1, est...
2     [Existe, algum, curso, para, cuidar, do, marke...
3     [Gostaria, de, renovar, antecipadamente, meu, ...
4     [O, vídeo, não, está, travando, as, vezes, no,...
5     [Que, trilha, vocês, recomendam, para, quem, q...
6                   [Quanto, custa, o, plano, premium?]
7     [Como, faço, para, postar, minha, dúvida, no, ...
8     [Já, trabalho, como, designer, e, queria, apre...
9     [Coloquei, meu, nome, com, letra, maiúscula, n...
10           [Onde, posso, acessar, meu, certificado?]
11    [Que, cursos, devo, fazer, para, crescer, na, ...
12    [Se, meu, acesso, expirar, ainda, tenho, acess...
13    [Achei, um, erro, no, vídeo, de, Excel., A, fó...
14    [Que, cursos, vocês, indicam, para, meu, filho...
15                [Posso, imprimir, meu, certificado?]
16    [Bom, dia,, tenho, interesse, na, parte, de, I...
```

```
17     [Qual, a, carreira, indicada, para, meu, filho?]
18    [Gostaria, de, saber, se, vocês, emitem, certi...
19               [qual, versão, do, laravel, no, curso]
20    [Olá,, quero, aprender, a, programar,, mas, nã...
21    [Comprei, o, plano, Premium, ontem, mas, Queri...
22    [Como, faço, para, me, tornar, um, moderador, ...
23    [estou, interessado, em, python, minha, duvida...
24    [Usei, o, cartão, do, meu, pai, e, ele, cancel...
25    [Como, faço, para, ser, convidado, a, me, torn...
26    [Me, cadastrei, com, o, email, antigo, do, pay...
27    [Quais, os, passos, para, me, tornar, um, dese...
28    [Tenho, o, plano, premiun, de, renovar, o, pla...
29    [Quero, ajudar, mais, a, comunidade, do, Alura...
30    [Terminei, a, faculdade, em, sistemas, de, inf...
31         [Além, do, paypal, posso, pagar, com, moip?]
32    [Encontrei, um, erro, de, digitação, em, uma, ...
33    [Posso, efetuar, o, pagamento, a, vista, via, ...
34    [Boa, noite,, equipe, do, Alura!, Senti, a, ne...
35    [Usei, a, conta, do, paypal, de, meus, pais, p...
36    [Desde, já,, desejo-te, um, ótimo, início, de,...
37    [Gostaria, de, saber, qual, o, melhor, curso, ...
38    [Posso, usar, o, cartão, de, meu, pai, para, e...
39    [Trabalhando, com, o, framework, Django, não, ...
40    [É, necessário, aprender, html,, css, e, javas...
41    [Estou, com, dificuldade, para, fazer, o, paga...
42    [Gostaria, de, saber, qual, o, melhor, caminho...
Name: email, dtype: object
```

Repare que agora cada linha refere-se a um array com cada uma das palavras separadas. Dado que temos vários arrays que contém todas as palavras de cada texto, o que precisamos fazer agora? Criar um único array que contenha todas essas palavras, ou seja, o nosso dicionário. Então começaremos criando um array chamado `dicionario`:

```
dicionario = []
```

Agora precisamos iterar sobre as listas da variável `textosQuebrados`:

```
for lista in textosQuebrados:
```

Então, para cada uma dessas listas, pediremos para estender-se ao nosso dicionário:

```
for lista in textosQuebrados:
  dicionario.extend(lista)
```

Agora vamos imprimir esse `dicionario`:

```
print(dicionario)
```

Vejamos o resultado da variável `dicionario`:

```
> python classificando_emails.py
```

```
['Se', 'eu', 'comprar', 'cinco', 'anos', 'antecipados,', 'eu', 'ganho', 'algum', 'desconto?', 'O',
...
'gosta', 'de', 'design', 'mas', 'n\xc3\xa3o', '\xc3\xa9', 'f\xc3\xa3', 'de', 'programa\xc3\xa7\xc3\
```

Agora temos um super dicionário! Entretanto observe a seguinte palavra: `programa\xc3\xa7\xc3\xa3o.` . Sabemos que essa palavra refere-se à programação, porém, o que aconteceu? Esse problema ocorreu justamente porque os caracteres "çã" não fazem parte do ASCII, porém, é apenas uma questão de impressão, pois todas as palavras foram inseridas corretamente. Além disso, existe um detalhe bem importante, que é a repetição de palavras, por exemplo, a palavra "com" ou "de" ou "a" se repetem! Como vimos anteriormente, o dicionário é um array de palavras **distintas**, ou seja, não podemos conter palavras repetidas dentro do nosso array.

Ao invés de utilizarmos uma lista, precisamos usar um conjunto, pois na matemática, dentro do estudo da teoria dos conjuntos, se temos os dados {1, 2, 3, 4, 5} não é permitido adicionar novamente qualquer número que esteja dentro desse conjunto, ou seja, podemos apenas **adicionar qualquer valor diferente** de {1, 2, 3, 4, 5}. Tecnicamente chamamos esses conjuntos de **set** que no inglês significa conjunto. Modificaremos o nosso dicionário para que ele seja um `set` :

```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.split(' ')
dicionario = set()

for lista in textosQuebrados:
    dicionario.extend(lista)

print dicionario
```

Então qual será o compormento da variável `dicionario` ? A cada vez que ele for adicionar um elemento dentro dele, ele vai verificar se já existe, caso não exista, ele adicionará, caso contrário, isto é, se existir, ele não adicionará. Além disso, precisamos modificar a forma que estamos adicionando os elementos, isto é, ao invés do `extends` , utilizaremos o `update` :

```
# restante do código

dicionario = set()

for lista in textosQuebrados:
    dicionario.update(lista)

print dicionario
```

Vamos rodar novamente e verificar o resultado:

```
> python classificando_emails.py
set(['', 'v\xc3\xaddeo', 'digita\xc3\xa7\xc3\xa3o', 'nenhum', 'vezes', 'quero', 'desconto?', 'Senti
```

Observe que agora, as palavras não se repetem! Portanto, para pegarmos um conjunto de elementos distintos, utilizamos o `set`. Porém, ainda existe um detalhe, pois quando tentamos procurar a palavra "como" dentro do nosso dicionário, ele acha tanto a palavra "como" quanto a palavra "Como". Em outras palavras, se uma palavra possui as mesmas letras, mas contém um acento diferente, ou então, uma pontuação ou letra maiúscula que diferencia uma da outra, será considerada uma palavra distinta. Então como podemos resolver esse problema? No caso, fazer com que palavras praticamente iguais, isto é, variando apenas entre letras maiúsculas ou minúsculas, não sejam adicionadas mais de uma vez? Quando nos deperâmetro com esse tipo de problema, precisamos fazer uma limpeza no nosso código, isto é, transformar todas as palavras em minúsculo ou tudo em maiúsculo para que, independemente se o usuário digitou maiúsculo ou minúsculo, as palavras sejam as mesmas. Então vem a questão:

- Se fosse um e-mail de *spam*? Faria sentido convertermos tudo para maiúsculo ou minúsculo?

Repare que, quando nos deparamos com esse tipo de situação, ter tanto as palavras maiúsculas quanto as minúsculas faz todo o sentido, pois podemos utilizar parâmetros como quantidade de palavras em maiúsculo ou minúsuculo, para avaliar se ele refere-se a um *spam* ou não. Atualmente não estamos interessados em realizar esse tipo de avaliação, mas e se o nosso objetivo é entender se o usuário está bravo? Com certeza quando nos deperarmos com textos em maiúsculo, saberemos que essa é a sensação desse usuário. Portanto, palavras em maiúsculo ou minúscolo, podem nos indicar alguma coisa, logo, precisamos avaliar se faz sentido ou não mantê-los, nesse caso, transformaremos todos em minúsculo. Mas como fazemos isso no código? Basta apenas, no momento em que transformamos a variável `textoPuros` em string, pedirmos que transforme todo o texto contido em minúsculo utilizando a função `lower`:

```
textosQuebrados = textosPuros.str.lower().str.split(' ')
```

Testando novamente o nosso arquivo `classificando_emails.py`, temos o seguinte resultado:

```
> python classificando_emails.py
set(['', 'v\xc3\xaddeo', 'digita\xc3\xa7\xc3\xa3o', 'nenhum', 'vezes', 'quero', 'desconto?', 'voc\xc
```

Note que, se procurarmos pela palavra "como" novamente, teremos apenas uma única palavra! Vejamos também a quantidade palavras que temos atualmente no nosso dicionário:

```
# restante do código

totalDePalavras = len(dicionario)

print dicionario
print totalDePalavras
```

Testando novamente:

```
> python classificando_emails.py
set(['', 'v\xc3\xaddeo', 'digita\xc3\xa7\xc3\xa3o', 'nenhum', 'vezes', 'quero', ... 'aulas,'])
365
```

Então podemos concluir que o nosso dicionário contém 365 palavras distintas! O que precisamos fazer agora? Atribuir um número para cada uma das palavras contidas no nosso dicionário, por exemplo, a primeira palavra será o número 0, a segunda o número 1, a terceira 2 e assim sucessivamente até chegar na última palavra. Nesse caso, os números irão variar de

0 a 364. Podemos utilizar a função `zip` do python que podemos dizer o que queremos do lado esquerdo e do lado direito, por exemplo, a primeira palavra do `dicionario` no lado esquerdo e o número 0 no lado direito, a segunda palavra do `dicionario` no lado esquerdo e o número 1 do lado direito. Primeiro, vamos chamar o `zip` enviando o nosso `dicionario` como primeiro parâmetro para que ele fique no lado esquerdo:

```
zip(dicionario)
```

Agora precisamos dizer o que queremos do lado direito, que são os números de 0 a 365 exclusive, ou seja 0 a 364. Isso significa que precisamos escrever todos os valores de 0 a 364? Não! Para resolver esse problema, utilizaremos a função `xrange`, que recebe um número como parâmetro e cria um intervalo de 0 até o número enviado por parâmetro **exclusive**, em outras palavras, se enviarmos a quantidade de elementos contidos no `dicionario`, o `xrange` fará o intervalo de 0 a 364. Então vamos adicionar o `xrange` como segundo parâmetro enviando a variável `totalDePalavras` como parâmetro que corresponde ao `len(dicionario)`:

```
zip(dicionario, xrange(totalDePalavras))
```

Agora, vamos excluir as impressões anteriores e vamos apenas imprimir o `zip`:

```
# restante do código

for lista in textosQuebrados:
    dicionario.update(lista)

totalDePalavras = len(dicionario)

print zip(dicionario, xrange(totalDePalavras))
```

Rodando o nosso arquivo `classificando_emails.py`:

```
> python classificando_emails.py
[('', 0), ('v\xc3\xaddeo', 1), ('digita\xc3\xa7\xc3\xa3o', 2), ('nenhum', 3), ('vezes', 4), ('quero'
```

Observe que agora cada elemento do array representa uma tupla (https://pt.wikipedia.org/wiki/Enupla), isto é, um par ordenado de uma palavra e um número. Portanto, cada palavra contida no `dicionario`, está associada a um número. Então agora, retornaremos o resultado da função `zip` para uma variável chamada `tuplas`:

```
# restante do código

for lista in textosQuebrados:
    dicionario.update(lista)

totalDePalavras = len(dicionario)
tuplas = zip(dicionario, xrange(totalDePalavras))
```

Porém, se queremos consultar o número de uma palavra, por exemplo, a palavra "pode", como fazemos isso? Vamos tentar pedindo para a `tuplas`. Nesse teste, utilizaremos o interpretador do python para facilitar os demais testes:

```
> python
>>> import pandas as pd
>>> classificacoes = pd.read_csv('emails.csv')
>>> textosPuros = classificacoes['email']
>>> textosQuebrados = textosPuros.str.lower().str.split(' ')
>>> dicionario = set()
>>>
>>> for lista in textosQuebrados:
...     dicionario.update(lista)
...
>>> totalDePalavras = len(dicionario)
>>> tuplas = zip(dicionario, xrange(totalDePalavras))
>>>
```

Tentaremos pedir pela palavra "pode":

```
>>> tuplas['pode']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not str
>>>
```

Uma tupla não permite a busca por meio dos valores, em outras palavras, acessamos os valores de uma tupla apenas por meio de suas posições, ou seja, pedindo por exemplo, pela posição 0, 1, 2 e assim sucessivamente:

```
>>> tuplas[0]
('', 0)
>>> tuplas[1]
('v\xc3\xaddeo', 1)
>>> tuplas[2]
('digita\xc3\xa7\xc3\xa3o', 2)
```

Lembre-se que para buscarmos algum elemento por meio do seu valor chave, precisamos utilizar o dicionário do python que é justamente um mapa que permite identificar um valor por meio de uma chave. Declaramos um mapa da seguinte forma:

```
>>> mapa = {}
```

Então vamos criar um dicionário chamado `palavrasEIndices` :

```
>>> mapa = {}
>>> palavrasEIndices = {}
```

Agora adicionaremos algumas palavras e suas respectivas posições:

```
>>> palavrasEIndices['pode']=15
>>> palavrasEIndices['poder']=18
```

Vejamos os valores contidos no mapa `palavrasEIndices` :

```
>>> print palavrasEIndices
{'pode': 15, 'poder': 18}
```

Note que se tentarmos buscar a posição do elemento, por meio da sua chave, por exemplo a palavra "pode":

```
>>> print palavrasEIndices['pode']
15
```

Considerando o que vimos sobre o dicionário do python, o que precisamos fazer agora? Transformar a variável `tuplas` em um dicionário, isto é, para cada elemento da variável `tuplas`, transformá-lo em um dicionário. Como fazemos isso? Simples! Faremos um for nas `tuplas`:

```
for palavra, indice in tuplas
```

Nesse instante estamos pegando cada `palavra` e `indice` dos elementos da `tuplas`, ou seja, a palavra e seu número. A partir deles, criaremos o nosso dicionário:

```
{palavra:indice for palavra, indice in tuplas}
```

Então atribuímos esse dicionário para a variável `palavrasEIndices`:

```
palavrasEIndices = {palavra:indice for palavra, indice in tuplas}
```

Vamos testar esse código, copie e cole no interpretador, e então, imprima o dicionário `palavrasEIndices`:

```
>>> palavrasEIndices = {palavra:indice for palavra, indice in tuplas}
>>> print palavrasEIndices
{'': 0, 'v\xc3\xaddeo': 1, 'digita\xc3\xa7\xc3\xa3o': 2, 'nenhum': 3, 'vezes': 4, 'quero': 5, 'desc
>>>
```

Repara que agora temos um dicionário com as nossas tuplas. Será que agora conseguimos encontrar a posição de alguma palavra buscando por ela mesmo? Por exemplo a palavra "pode"? Vamos testar!

```
>>> print palavrasEIndices['pode']
361
```

Agora temos um mapa que indica se existe uma palavra no nosso dicionário e qual é a posição dela, por exemplo, vamos tentar a palavra "guilherme":

```
>>> print palavrasEIndices['guilherme']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'guilherme'
```

Perceba que quando não existir uma palavra, ele nos devolverá essa mensagem, indicando um erro de chave, pois não existe para esse mapa, porém, se procurarmos palavras que existam, como por exemplo, "com", "como":

```
>>> print palavrasEIndices['com']
226
>>> print palavrasEIndices['como']
215
```

Ele nos devolve suas respectivas posições. O que falta agora para o nosso algoritmo? É justamente varrer cada texto e marcar quantas vezes uma deternada palavra se repete, por exemplo, considere o texto abaixo:

- Se eu comprar cinco anos antecipados, eu ganho algum desconto?

O nosso algoritmo deverá verificar primeiro a palavra "Se", então ele vai encontrar a posição dessa palavra e vai marcar como 1, depois vai verificar a posição da palavra "eu", e então, marcará como 1 e assim por diante. Em outras palavras, precisamos pegar os nossos textos e transformá-los em arrays de números por meio do nosso dicionário que fará o papel de tradutor, isto é, traduzir um texto em array de números. Portanto, podemos chamá-lo de `tradutor`:

```
# restante do código

tuplas = zip(dicionario, xrange(totalDePalavras))

tradutor = {palavra:indice for palavra, indice in tuplas}
```

Antes de contabilizar a quantidade de vezes que uma palavra se repete nos nossos textos, vamos verificar as palavras contidas no primeiro texto da variável `textosQuebrados`:

```
>>> print textosQuebrados[0]
['se', 'eu', 'comprar', 'cinco', 'anos', 'antecipados,', 'eu', 'ganho', 'algum', 'desconto?']
```

Como podemos fazer para analisar cada uma dessas palavras? Precisamos fazer um laço que passe por cada uma dessas palavras, ou seja, um `for`:

```
for palavra in textosQuebrados[0]:
```

E então, vamos imprimir cada uma dessas palavras:

```
>>> for palavra in textosQuebrados[0]:
...   print palavra
...
se
eu
comprar
cinco
anos
antecipados,
eu
ganho
```

```
algum
desconto?
```

Qual é o próximo passo? Nesse exemplo, precisamos pegar primeiro a palavra "se" e contabilizar 1 no array que representa essa frase, porém, ainda não temos esse array, logo, precisamos criá-lo. Lembra que, quando tinhamos apenas aquelas 3 frases de exemplo:

- Se eu comprar cinco anos antecipados, eu ganho algum desconto?
- Eu ganho desconto se comprar cinco anos antecipados?
- Ao terminar um curso, eu ganho um certificado?

Tinhamos o seguinte dicionário que suportava todas as essas frases:

```
[Se, eu, comprar, cinco, anos, antecipados, ganho, algum, desconto, Ao, terminar, um, curso, certifi
```

Note que esse dicionário contém apenas 14 palavras distintas, portanto, para essas frases acima, utilizamos um array de tamanho 14 para todas elas. Entretanto, no nosso cenário atual, temos 365 palavras distintas! Em outras palavras, um array que representa qualquer frase dos nossos textos precisará ter um tamanho de 365, isto é, 365 posições. Portanto, criaremos um vetor com 365 posições, isto é, um vetor que tenha o tamanho da variável `totalDePalavras` e que iniciem com valor 0:

```
vetor = [0] * totalDePalavras
```

Vamos verificar se o nosso vetor está funcionando como o esperado:

```
>>> vetor = [0] * totalDePalavras
>>> print vetor
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
>>> print len(vetor)
365
```

Observe que temos um vetor de tamanho 365 com todos os valores igual a 0. Essa é a representação de uma frase vazia, isto é, que não tenha pelo menos uma posição com valor acima de 0. Agora que temos o nosso array que permite representarmos todas as frases do nosso universo, ou seja, todas as frases que são escritas com as palavras contidas do nosso dicionário, precisamos verificar quantas vezes uma palavra se repete. Vamos utilizar o exemplo anterior:

```
>>> for palavra in textosQuebrados[0]:
...   print palavra
...
se
eu
comprar
cinco
anos
antecipados,
eu
ganho
algum
desconto?
```

Então qual é o primeiro passo? Pegaremos a palavra "se", em seguida, precisamos verificar se ela existe no nosso dicionário, isto é, na variável `tradutor`, e então, caso exista, pegamos a sua posição, por exemplo, se a palavra "se" estiver na posição 32, pegamos o array que representa a frase e somamos 1 na posição 32. Precisamos implementar esses passos dentro do nosso código, porém, antes de começarmos com essa implementação, adicionaremos todos os passos realizados no interpretador do python dentro do nosso arquivo `classificando_emails.py`, o nosso código atual fica da seguinte forma:

```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.lower().str.split(' ')
dicionario = set()

for lista in textosQuebrados:
    dicionario.update(lista)

totalDePalavras = len(dicionario)
tuplas = zip(dicionario, xrange(totalDePalavras))
tradutor = {palavra:indice for palavra, indice in tuplas}
print totalDePalavras

texto = textosQuebrados[0]
vetor = [0] * totalDePalavras

print texto
print vetor
for palavra in texto:
    print palavra
```

Qual é o nosso primeiro passo? Precisamos verificar palavra por palavra, portanto, iremos trabalhar na instrução `for` que, a partir de um texto extrai, uma palavra:

```
for palavra in texto:
    print palavra
```

Dada uma palavra do nosso texto, precisamos verificar se ela existe no nosso dicionário, ou seja, precisamos adicionar um `if` que vai verificar se ela está contida no nosso dicionário, nesse caso, a variável `tradutor`:

```
for palavra in texto:
  if palavra in tradutor:
    print palavra
```

Caso a palavra exista no nosso dicionário, o que precisamos fazer? Imprima a palavra? Não! Então vamos retirar o código que imprime a palavra:

```
for palavra in texto:
    if palavra in tradutor:
```

E agora? O que temos que fazer? Precisamos somar 1 no `vetor` de acordo com o posição da palavra, mas como saberemos a posição? Simples! Basta apenas pedir a posição da palavra deseja para o nosso dicionário `tradutor` que tem o objetivo de traduzir uma palavra contida no dicionário em número, isto é, sua posição:

```
for palavra in texto:
    if palavra in tradutor:
        posicao = tradutor[palavra]
```

Em seguida, somamos 1 na variável `vetor` de acordo com a posição contida na variável `posicao` :

```
for palavra in texto:
    if palavra in tradutor:
        posicao = tradutor[palavra]
        vetor[posicao] += 1
```

Esse nosso código está fazendo os seguintes passos:

- Pega uma palavra contida na nossa variável `texto` .
- Verifica se essa palavra está contida no dicionário `tradutor` .
- Se sim, pede para o tradutor devolve a posição, e então, somamos 1 no `vetor` .

Ele vai repetindo esse processo até verificar todas as palavras contidas na variável `texto` que é um array de strings. Por fim, o nosso resultado final é justamente a variável `vetor` , pois ela traduz o texto em um array de número. Ao invés de imprimir a variável `vetor` antes do `for` , vamos imprimí-la por último, pois queremos verificar a nossa palavra traduzida:

```
for palavra in texto:
    if palavra in tradutor:
        posicao = tradutor[palavra]
        vetor[posicao] += 1

print vetor
```

Vamos testar o nosso arquivo `classificando_emails.py` e verificar o resultado:

```
> python classificando_emails.py
365
['se', 'eu', 'comprar', 'cinco', 'anos', 'antecipados,', 'eu', 'ganho', 'algum', 'desconto?']
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Observe que foi impresso o array com algumas posições preenchidas. As palavras contidas nesse texto, apareceram apenas uma vez com exceção de uma que foram duas vezes, nesse caso, a palavra "eu", pois se repete. Note que esse array foi impresso pois a palavra que pedimos para ele traduzir contém palavras que o nosso dicionário, ou seja, a variável `tradutor` conhece, caso o texto, não tivesse nenhuma das palavras do nosso `tradutor` , ele simplesmente devolveria um array preenchido com zeros apenas, pois ele iria ignorar palavra por palavra dentro desse texto. Aparentemente terminamos os

passos que precisávamos realizar, porém, temos que reutilizar esse processo, em outras palavras, precisamos transformar esse `for` que faz todo esse trabalho de, a partir de um texto, pegar uma palavra e transformá-la em um array de números, em uma função para que possamos traduzir todos os nossos textos. Então criaremos uma função que se chamará `vetorizar_texto()` :

```
# restante do código

def vetorizar_texto():

texto = textosQuebrados[0]
vetor = [0] * totalDePalavras

print texto

for palavra in texto:
    if palavra in tradutor:
        posicao = tradutor[palavra]
        vetor[posicao] += 1

print vetor
```

Precisamos enviar tanto o texto quanto o tradutor que irá traduzir o texto:

```
def vetorizar_texto(texto, tradutor):
```

Dentro dessa função o nosso primeiro passo é calcular o tamanho do vetor que irá representará cada palavra contida no texto, podemos reutilizar o código que fizemos anteriormente:

```
def vetorizar_texto(texto, tradutor):
    vetor = [0] * totalDePalavras
```

Porém, perceba que não estamos recebendo a variável `totalDePalavras` , ou seja, precisamos recebê-la como parâmetro. Entretanto, precisamos realmente de uma variável para saber a quantidade total de palavras contidas no nosso `tradutor` ? Não! Pois podemos pegar o próprio tamanho do `tradutor` que corresponde à quantidade total de palavras, isto é, a quantidade exata que o nosso `vetor` precisa ter:

```
def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
```

Agora podemos copiar e colar todo o `for` utilizado pra iterar sobre cada palavra contida em um texto:

```
def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1
```

Por fim, precisamos retornar o vetor obtido:

```
def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor
```

Antes de rodarmos o nosso código, ao invés de imprimir a variável `vetor`, vamos imprimir o retorno da função `vetorizar_texto`:

```
# restante do código

def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor

texto = textosQuebrados[0]

print texto
print vetorizar_texto(texto, tradutor)
```

Vamos testar o nosso código e verificar o resultado:

```
> python classificando_emails.py
365
['se', 'eu', 'comprar', 'cinco', 'anos', 'antecipados,', 'eu', 'ganho', 'algum', 'desconto?']
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Veja que a nossa função devolve o nosso texto vetorizado da mesma forma que antes. Vamos testar outros textos para verificar seus vetores? Vejamos como fica:

```
def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor

print vetorizar_texto(textosQuebrados[0], tradutor)
print vetorizar_texto(textosQuebrados[1], tradutor)
print vetorizar_texto(textosQuebrados[2], tradutor)
```

Testando novamente o nosso arquivo `classificando_emails.py`, temos o seguinte resultado:

```
> python classificando_emails.py
365
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
```

O nosso algoritmo imprime um vetor diferente para cada texto distinto que temos. Porém, precisamos realizar essa tarefa para todos os textos que temos dentro da variável `textosQuebrados`, como faremos isso? Fazemos um `for` para pegar cada texto contido na variável `textosQuebrados`:

```
# restante do código

for texto in textosQuebrados
```

Em seguida, aplicamos a função `vetorizar_texto()`:

```
# restante do código

vetorizar_texto(texto, tradutor) for texto in textosQuebrados
```

Entretanto, precisamos transformar cada resultado desse em um array, portanto, faremos isso:

```
# restante do código

[vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
```

Esses serão os nossos vetores de textos, ou melhor, atribuíremos esses arrays para a variável `vetoresDeTexto`:

```
# restante do código

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
```

Vamos rodar o nosso código, porém, apague as 3 linhas que utilizamos para testar a chamada de outros texto para a nossa função e imprima apenas a variável `vetoresDeTexto`:

```
# restante do código

def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor
```

```
vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
print vetoresDeTexto
```

Testando novamente o nosso arquivo `classificando_emails.py` :

```
[[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Ele imprime um array de arrays bem gigante. Nesse caso é um array que contém 43 arrays sendo que cada array contém 365 posições que representa cada um dos textos vetorizados. Perceba que agora concluímos o nosso primeiro objetivo que é justamente pegar todos os textos que temos e transformá-los em arrays númericos do mesmo tamanho para todos. Então qual é o próximo passo? Utilizar o nosso algoritmo que é capaz de realizar as classificações. Entretanto, o que o nosso algoritmo precisa para classificar? Dos dados e das marcações, os nossos dados estão contidos na variável `vetoresDeTexto` mas e as nossas marcações? A qual coluna do nosso arquivo `emails.csv` refere-se às marcações para os nossos textos? Vejamos:

```
email,classificacao
"Se eu comprar cinco anos antecipados, eu ganho algum desconto?",1
O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?,2
Existe algum curso para cuidar do marketing da minha empresa?,3
"Gostaria de renovar antecipadamente meu plano, como posso fazer?",1
O vídeo não está travando as vezes no meu navegador. Como trocar o player?,2
Que trilha vocês recomendam para quem quer começar com programação?,3
...
```

Observe que temos a coluna `email` e `classificacao` , ou seja, as nossas marcações para os nossos textos são todos os valores contidos na coluna `classificacao` , logo, precisamos pedir para o nosso datagrama `classificacoes` :

```
import pandas as pd
classificacoes = pd.read_csv('emails.csv')

# restante do código
```

Os valores da coluna `classificacao` e atribuímos para a variável `marcas` :

```
# restante do código

def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']
```

A partir de agora precisamos pegar tanto o `X` quanto o `Y`, lembra que o `X` refere-se aos nossos dados? E, o `Y`, às marcações? Em outras palavras, atribuiremos a variável `vetoresDeTexto` ao `X` e as `marcas` para o `Y`:

```
# restante do código

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']

X = vetoresDeTexto
Y = marcas
```

No nosso algoritmo de classificação, após coletarmos o `X` e `Y`, definíamos os percentuais de treino, nesse caso utilizaremos 80%:

```
# restante do código

X = vetoresDeTexto
Y = marcas

porcentagem_de_treino = 0.8
```

Esses 80% significam treino e teste, portanto, os 20% restantes serão utilizados para validação. Agora precisamos definir o tamanho do treino e teste, e também, o tamanho de validação. Para o tamanho de treino e teste, multiplicaremos a variável `porcentagem_de_treino` com o tamanho do `Y`:

```
# restante do código

X = vetoresDeTexto
Y = marcas

porcentagem_de_treino = 0.8

tamanho_de_treino = porcentagem_de_treino * len(Y)
```

Porém, o tamanho de validação será o tamanho do `Y` menos a variável `tamanho_de_treino`:

```
# restante do código

X = vetoresDeTexto
Y = marcas

porcentagem_de_treino = 0.8

tamanho_de_treino = porcentagem_de_treino * len(Y)
tamanho_de_validacao = len(Y) - tamanho_de_treino
```

O nosso próximo passo é justamente separar os dados e marcacações de treino com os dados e marcações de validação. Então começaremos pelos dados de treino:

```
# restante do código

treino_dados = X[0:tamanho_de_treino]
treino_marcacoes = Y[0:tamanho_de_treino]
```

Para os dados e marcações de teste, pegaremos desde o primeiro elemento (posição 0) até a variável `tamanho_de_treino`. E para os dados e marcações de validação? Pegaremos a partir da variável `tamanho_de_treino` até em diante:

```
validacao_dados = X[tamanho_de_treino:]
validacao_marcacoes = Y[tamanho_de_treino:]
```

Definimos todos os nosso dados que utilizaremos para os nossos algoritmos, qual era o próximo passo mesmo? Realizar o `fit e predict`, em outras palavras, realizar a mesma tarefa que fazíamos anteriormente nos nossos algoritmos anteriores, então vamos dar uma olhada no nosso arquivo `situacao_do_cliente_kfold.py`:

```
import pandas as pd
from collections import Counter
import numpy as np
from sklearn.cross_validation import cross_val_score

df = pd.read_csv('situacao_do_cliente.csv')
X_df = df[['recencia','frequencia', 'semanas_de_inscricao']]
Y_df = df['situacao']

Xdummies_df = pd.get_dummies(X_df).astype(int)
Ydummies_df = Y_df

X = Xdummies_df.values
Y = Ydummies_df.values

porcentagem_de_treino = 0.9

tamanho_de_treino = porcentagem_de_treino * len(Y)

treino_dados = X[:tamanho_de_treino]
treino_marcacoes = Y[:tamanho_de_treino]

validacao_dados = X[tamanho_de_treino:]
validacao_marcacoes = Y[tamanho_de_treino:]


def fit_and_predict(nome, modelo, treino_dados, treino_marcacoes):
    k = 10
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
    taxa_de_acerto = np.mean(scores)
    msg = "Taxa de acerto do {0}: {1}".format(nome, taxa_de_acerto)
    print msg
    return taxa_de_acerto

def teste_real(modelo, validacao_dados, validacao_marcacoes):
    resultado = modelo.predict(validacao_dados)

    acertos = resultado == validacao_marcacoes
```

```
        total_de_acertos = sum(acertos)
        total_de_elementos = len(validacao_marcacoes)

        taxa_de_acerto = 100.0 * total_de_acertos / total_de_elementos

        msg = "Taxa de acerto do vencedor entre os dois algoritmos no mundo real: {0}".format(taxa_de_ac
        print(msg)

    resultados = {}

    from sklearn.multiclass import OneVsRestClassifier
    from sklearn.svm import LinearSVC
    modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
    resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
    resultados[resultadoOneVsRest] = modeloOneVsRest

    from sklearn.multiclass import OneVsOneClassifier
    modeloOneVsOne = OneVsOneClassifier(LinearSVC(random_state = 0))
    resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
    resultados[resultadoOneVsOne] = modeloOneVsOne

    from sklearn.naive_bayes import MultinomialNB
    modeloMultinomial = MultinomialNB()
    resultadoMultinomial = fit_and_predict("MultinomialNB", modeloMultinomial, treino_dados, treino_marc
    resultados[resultadoMultinomial] = modeloMultinomial

    from sklearn.ensemble import AdaBoostClassifier
    modeloAdaBoost = AdaBoostClassifier()
    resultadoAdaBoost = fit_and_predict("AdaBoostClassifier", modeloAdaBoost, treino_dados, treino_marca
    resultados[resultadoAdaBoost] = modeloAdaBoost

    print resultados

    maximo = max(resultados)
    vencedor = resultados[maximo]

    print "Vencerdor: "
    print vencedor

    vencedor.fit(treino_dados, treino_marcacoes)

    teste_real(vencedor, validacao_dados, validacao_marcacoes)

    acerto_base = max(Counter(validacao_marcacoes).itervalues())
    taxa_de_acerto_base = 100.0 * acerto_base / len(validacao_marcacoes)
    print("Taxa de acerto base: %f" % taxa_de_acerto_base)

    total_de_elementos = len(validacao_dados)
    print("Total de teste: %d" % total_de_elementos)
```

Veja que realizamos todos os passos inicias da mesma forma que foi implementado no `situacao_do_cliente_kfold.py`, portanto, vamos pegar as funções adiante. Começaremos pelo `fit_and_predict`. Copie e cole no arquivo `classificando_emails.py`. O nosso arquivo fica com o seguinte código:

```
#!-*- coding: utf8 -*-
texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

import pandas as pd
classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.lower().str.split(' ')
dicionario = set()

for lista in textosQuebrados:
    dicionario.update(lista)

totalDePalavras = len(dicionario)
tuplas = zip(dicionario, xrange(totalDePalavras))
tradutor = {palavra:indice for palavra, indice in tuplas}
print totalDePalavras

def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']

X = vetoresDeTexto
Y = marcas

porcentagem_de_treino = 0.8

tamanho_de_treino = porcentagem_de_treino * len(Y)
tamanho_de_validacao = len(Y) - tamanho_de_treino

treino_dados = X[0:tamanho_de_treino]
treino_marcacoes = Y[0:tamanho_de_treino]

validacao_dados = X[tamanho_de_treino:]
validacao_marcacoes = Y[tamanho_de_treino:]

def fit_and_predict(nome, modelo, treino_dados, treino_marcacoes):
    k = 10
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
    taxa_de_acerto = np.mean(scores)
    msg = "Taxa de acerto do {0}: {1}".format(nome, taxa_de_acerto)
    print msg
    return taxa_de_acerto
```

Mas ainda precisamos de um algoritmo para testar o nosso código. Utilizaremos o mesmo exemplo do `OneVsRest` no arquivo `situacao_do_cliente_kfold.py` , isto é, copie e cole no arquivo `classificando_emails.py` :

```
# restante do código

def fit_and_predict(nome, modelo, treino_dados, treino_marcacoes):
    k = 10
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
    taxa_de_acerto = np.mean(scores)
    msg = "Taxa de acerto do {0}: {1}".format(nome, taxa_de_acerto)
    print msg
    return taxa_de_acerto

resultados = {}

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
resultados[resultadoOneVsRest] = modeloOneVsRest
```

Vamos rodar o nosso arquivo `classificando_emails.py` e verificar qual é o resultado para o algoritmo `OneVsRest`:

```
> python classificando_emails.py
365
Traceback (most recent call last):
  File "classificando_emails.py", line 40, in <module>
    treino_dados = X[0:tamanho_de_treino]
TypeError: slice indices must be integers or None or have an __index__ method
```

Observe que ele disse que um dos índices quando estamos pegando os dados de treino não refere-se a um número inteiro. Sabemos que 0 é um número inteiro, portanto, vamos investigar a variável `tamanho_de_treino` realizando uma simples impressão:

```
# restante do código

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']

X = vetoresDeTexto
Y = marcas

porcentagem_de_treino = 0.8

tamanho_de_treino = porcentagem_de_treino * len(Y)
tamanho_de_validacao = len(Y) - tamanho_de_treino

print tamanho_de_treino
```

Rodando novamente o arquivo `classificando_emails.py`:

```
> python classificando_emails.py
365
34.4
```

```
Traceback (most recent call last):
  File "classificando_emails.py", line 42, in <module>
    treino_dados = X[0:tamanho_de_treino]
TypeError: slice indices must be integers or None or have an __index__ method
```

Veja que o número retornado foi 34.4, pois 80% de 43 é 34,40%, ou seja, um número decimal, portanto, precisamos arredondá-lo. Para isso, basta informamos que o resultado da operação que multiplica a variável `porcentagem_de_treino` com o tamanho do `Y` seja um número inteiro:

```
# restante do código

tamanho_de_treino = int(porcentagem_de_treino * len(Y))
tamanho_de_validacao = len(Y) - tamanho_de_treino
```

Testando novamente o nosso algoritmo:

```
> python classificando_emails.py
365
34
Traceback (most recent call last):
  File "classificando_emails.py", line 61, in <module>
    resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoe
  File "classificando_emails.py", line 50, in fit_and_predict
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
NameError: global name 'cross_val_score' is not defined
```

Ele apresenta um novo erro. Percebe que é de extrema importância rodarmos passo a passo durante o processo de implementação? Quando aderimos a essa prática, temos um maior controle em detectar as falhas do nosso algoritmo, por exemplo, imagine que pegássemos todo o código do arquivo `situacao_do_cliente_kfold.py` e começasse a dar erro em diversos pontos, seria um pesadelo ficar verificando todos os pontos ao mesmo tempo. É válido lembrar que, se tivermos utilizando dados muito similares aos quais utilizávamos anteriormente, provavelmente não haverá problema algum copiar e colar todo o código. Então vamos concertar esse erro que está sendo apresentado. Observe a mensagem:

```
NameError: global name 'cross_val_score' is not defined
```

Isso significa que não importamos a função `cross_val_score`, então vamos importá-la! Entretanto, aproveitando que iremos importar essa função, vamos verificar os demais imports contidos no arquivo `situacao_do_cliente_kfold.py`, pois, provavelmente serão solicitados:

```
import pandas as pd
from collections import Counter
import numpy as np
from sklearn.cross_validation import cross_val_score

# restante do código
```

Dentro do arquivo `classificando_emails.py`, utilizaremos os mesmos imports:

```
#!-*- coding: utf8 -*-

import pandas as pd
from collections import Counter
import numpy as np
from sklearn.cross_validation import cross_val_score

texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.lower().str.split(' ')
dicionario = set()

# restante do código
```

Note que precisamos adicionar esses imports logo depois do coding. Agora que realizamos todos os imports, vamos testar o nosso algoritmo e verificar o resultado:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
```

Agora sim o nosso algoritmo funcionou! Observe que a taxa de acerto foi de 72,33% no teste, ou seja, em 72,33% das vezes ele saberá encaminhar para o lugar certo o e-mail recebido. É válido lembrar que precisamos realizar todo o processo que fazíamos antes, ou seja, comparar com o algoritmo base, e também, realizar o teste de validação. Logo mais implementaremos todos esses passos, mas, vamos adicionar os demais algoritmos, o próximo que implementaremos será o `OneVsOne`. Copiaremos e colaremos o algoritmo a partir do arquivo `situacao_do_cliente_kfold.py`:

```
# restante do código

resultados = {}

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
resultados[resultadoOneVsRest] = modeloOneVsRest

from sklearn.multiclass import OneVsOneClassifier
modeloOneVsOne = OneVsOneClassifier(LinearSVC(random_state = 0))
resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
resultados[resultadoOneVsOne] = modeloOneVsOne
```

Vamos rodar e verificar o resultado:

```
> python classificando_emails.py
365
34
```

```
34
Taxa de acerto do OneVsRest: 0.723333333333
Traceback (most recent call last):
  File "classificando_emails.py", line 71, in <module>
    resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
  File "classificando_emails.py", line 55, in fit_and_predict
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/cross_validation.py", line 1433, in cross_va

    for train, test in cv)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 800, in
    while self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 658, in
    self._dispatch(tasks)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 566, in
    job = ImmediateComputeBatch(batch)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 180, in
    self.results = batch()
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 72, in __
    return [func(*args, **kwargs) for func, args, kwargs in self.items]
  File "/usr/local/lib/python2.7/dist-packages/sklearn/cross_validation.py", line 1531, in _fit_and_
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/multiclass.py", line 517, in fit
    for i in range(n_classes) for j in range(i + 1, n_classes))
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 800, in
    while self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 658, in
    self._dispatch(tasks)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 566, in
    job = ImmediateComputeBatch(batch)
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 180, in
    self.results = batch()
  File "/usr/local/lib/python2.7/dist-packages/sklearn/externals/joblib/parallel.py", line 72, in __
    return [func(*args, **kwargs) for func, args, kwargs in self.items]
  File "/usr/local/lib/python2.7/dist-packages/sklearn/multiclass.py", line 426, in _fit_ovo_binary
    ind = np.arange(X.shape[0])
AttributeError: 'list' object has no attribute 'shape'
```

Novamente um erro quando tentamos adicionar um outro algoritmo. Provavelmente está acontecendo algum passo no processo de transformar o array de string em números. Repare que a mensagem de erro nos informa que o objeto "list" não tem 'shape', isso significa que os objetos que estamos passando para o algoritmo `OneVsOne` não permitem ser trabalhados conforme o `OneVsOne` funciona. Precisamos verificar que tipo de array estamos passando para o nosso algoritmo. Atualmente, estamos enviando 2 valores para o nosso algoritmo:

```
X = vetoresDeTexto
Y = marcas
```

Na variável `vetoresDeTexto` temos uma lista e na variável `marcas` um data frame. Para transformarmos a lista `vetoresDeTexto` em um array do python, podemos utilizar a função `array` do `numpy`:

```
# restante do código

X = np.array(vetoresDeTexto)
Y = marcas
```

E as `marcas` ? Como podemos transformá-las em um array? Precisamos fazer um processo um pouco diferente, pois ele é um data frame, ou seja, primeiro precisaremos transformá-lo em uma lista:

```
# restante do código

X = np.array(vetoresDeTexto)
Y = marcas.tolist()
```

Em seguida, podemos utilizar o `numpy` para transformar a lista em um array:

```
# restante do código

X = np.array(vetoresDeTexto)
Y = np.array(marcas.tolist())
```

Agora que temos 2 arrays, vamos testar o nosso algoritmo:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
Taxa de acerto do OneVsOne: 0.656666666667
```

Repare que agora o nosso código funcionou, resultando em 72,33% para o `OneVsRest` e 65,66% para o `OneVsOne` . Então vamos para o próximo algoritmo, ou seja, o `Multinomial` :

```
# restante do código

resultados = {}

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
resultados[resultadoOneVsRest] = modeloOneVsRest

from sklearn.multiclass import OneVsOneClassifier
modeloOneVsOne = OneVsOneClassifier(LinearSVC(random_state = 0))
resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
resultados[resultadoOneVsOne] = modeloOneVsOne

from sklearn.naive_bayes import MultinomialNB
modeloMultinomial = MultinomialNB()
resultadoMultinomial = fit_and_predict("MultinomialNB", modeloMultinomial, treino_dados, treino_marc
resultados[resultadoMultinomial] = modeloMultinomial
```

Vejamos o resultado:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
Taxa de acerto do OneVsOne: 0.656666666667
Taxa de acerto do MultinomialNB: 0.69
```

Rodou sem nenhum problema! Por fim, adicionaremos o `AdaBoost` :

```
# restante do código

resultados = {}

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
resultados[resultadoOneVsRest] = modeloOneVsRest

from sklearn.multiclass import OneVsOneClassifier
modeloOneVsOne = OneVsOneClassifier(LinearSVC(random_state = 0))
resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
resultados[resultadoOneVsOne] = modeloOneVsOne

from sklearn.naive_bayes import MultinomialNB
modeloMultinomial = MultinomialNB()
resultadoMultinomial = fit_and_predict("MultinomialNB", modeloMultinomial, treino_dados, treino_marc
resultados[resultadoMultinomial] = modeloMultinomial

from sklearn.ensemble import AdaBoostClassifier
modeloAdaBoost = AdaBoostClassifier()
resultadoAdaBoost = fit_and_predict("AdaBoostClassifier", modeloAdaBoost, treino_dados, treino_marca
resultados[resultadoAdaBoost] = modeloAdaBoost
```

Vejamos o resultado:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
Taxa de acerto do OneVsOne: 0.656666666667
Taxa de acerto do MultinomialNB: 0.69
Taxa de acerto do AdaBoostClassifier: 0.473333333333
```

Todos os algoritmo estão funcionando, o que está faltando agora? Adicionar os demais passos, isto é, em que elegemos o algoritmo vencedor, rodamos o teste do mundo real, e então, rodamos o algoritmo base e verificamos o resultado final. Então vamos adicionar esse código:

```
# restante do código

print resultados
```

```
maximo = max(resultados)
vencedor = resultados[maximo]

print "Vencerdor: "
print vencedor

vencedor.fit(treino_dados, treino_marcacoes)

teste_real(vencedor, validacao_dados, validacao_marcacoes)

acerto_base = max(Counter(validacao_marcacoes).itervalues())
taxa_de_acerto_base = 100.0 * acerto_base / len(validacao_marcacoes)
print("Taxa de acerto base: %f" % taxa_de_acerto_base)

total_de_elementos = len(validacao_dados)
print("Total de teste: %d" % total_de_elementos)
```

Note que estamos utilizando a função `teste_real` , portanto, vamos adicioná-la também:

```
# restante do código

def fit_and_predict(nome, modelo, treino_dados, treino_marcacoes):
    k = 10
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
    taxa_de_acerto = np.mean(scores)
    msg = "Taxa de acerto do {0}: {1}".format(nome, taxa_de_acerto)
    print msg
    return taxa_de_acerto

def teste_real(modelo, validacao_dados, validacao_marcacoes):
    resultado = modelo.predict(validacao_dados)

    acertos = resultado == validacao_marcacoes

    total_de_acertos = sum(acertos)
    total_de_elementos = len(validacao_marcacoes)

    taxa_de_acerto = 100.0 * total_de_acertos / total_de_elementos


    msg = "Taxa de acerto do vencedor entre os dois algoritmos no mundo real: {0}".format(taxa_de_ac
    print(msg)
```

Testando o nosso código:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
Taxa de acerto do OneVsOne: 0.656666666667
Taxa de acerto do MultinomialNB: 0.69
Taxa de acerto do AdaBoostClassifier: 0.423333333333
{0.65666666666666651: OneVsOneClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000
```

```
        intercept_scaling=1, loss='squared_hinge', max_iter=1000,
        multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
        verbose=0),
             n_jobs=1), 0.72333333333333338: OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weigh
        intercept_scaling=1, loss='squared_hinge', max_iter=1000,
        multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
        verbose=0),
             n_jobs=1), 0.68999999999999995: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
             learning_rate=1.0, n_estimators=50, random_state=None)}
Vencerdor:
OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
     verbose=0),

          n_jobs=1)
Taxa de acerto do vencedor entre os dois algoritmos no mundo real: 88.8888888889
Taxa de acerto base: 44.444444
Total de teste: 9
```

Está funcionando conforme o esperado. O vencedor foi o `OneVsRest` com 72,33% acertando 88,88% das vezes no teste de validação. Note que o algoritmo base teve um resultado de 44,44%. Então podemos concluir que, se utilizássemos o algoritmo base, em 56% das vezes, teríamos que classificar manualmente os e-mails, mas, se utilizássemos o `OneVsRest`, em apenas 11% das vezes faríamos essa tarefa. Além disso, repara o valor que o `AdaBoost` nos apresentou dessa vez:

```
Taxa de acerto do AdaBoostClassifier: 0.423333333333
```

Anteriormente, ele havia nos apresentado 47,33% de taxa de acerto, mas dessa vez ele nos apresentou 42,33% por que será que isso aconteceu? O algoritmo `AdaBoostClassifier` tenta adaptar o conjunto de dados que ele recebe para encontrar o melhor resultado e, nesse processo, ele utiliza cópias dos dados para realizar essas tarefas, porém, esses processos são baseados em parâmetros aleatórios.

A aleatoriedade significa que a cada nova execução podemos ter um resultado diferente!

Como vimos anteriormente, quando estamos utilizando algoritmos classificadores, precisamos que os resultados sejam fixos, pois se não, teremos olhado diversas vezes os mesmos dados, viciando nossa decisão, ou seja, não poderemos tomar alguma decisão concreta sabendo que os resultados podem acontecer por "sorte" dessa vez ou "azar da próxima". Precisamos também que a execução possa ser repetida da mesma maneira que antes, se desejamos atualizar com novas partes o nosso programa. Como podemos resolver isso?

Em algoritmos que se baseiam em valores aleatórios, existe um parâmetro, geralmente chamado seed (https://en.wikipedia.org/wiki/Random_seed), que é um número de inicialização para a geração dos números aleatórios, ou seja, por meio desse número, os demais números "aleatórios" são gerados, por exemplo, se o valor do seed for fixo, significa que os números gerados a partir dele serão sempre os mesmos, porém, caso o contrário, isto é, se o seed for sempre um valor diferente a cada vez que rodarmos, haverá a possibilidade de termos números diferentes aos anteriores, no caso do `AdaBoost`, quando não informamos o valor do seed, a cada instância ele utiliza um valor diferente, é justamente por esse motivo que tivemos resultados diferentes. Em outras palavras, para resolver esse problema, basta apenas informarmos um valor fixo para o seed por meio do parâmetro `random_state` no momento em que criamos o `AdaBoostClassifier`. Utilizaremos o valor 0:

```
# restante do código
```

```
from sklearn.ensemble import AdaBoostClassifier
modeloAdaBoost = AdaBoostClassifier(random_state=0)
resultadoAdaBoost = fit_and_predict("AdaBoostClassifier", modeloAdaBoost, treino_dados, treino_marca
resultados[resultadoAdaBoost] = modeloAdaBoost
```

Vamos rodar mais uma vez o nosso algoritmo e verificar o resultado:

```
> python classificando_emails.py
365
34
Taxa de acerto do OneVsRest: 0.723333333333
Taxa de acerto do OneVsOne: 0.656666666667
Taxa de acerto do MultinomialNB: 0.69
Taxa de acerto do AdaBoostClassifier: 0.423333333333
{0.65666666666666651: OneVsOneClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fi
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
     verbose=0),
          n_jobs=1), 0.72333333333333338: OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weigh
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
     verbose=0),
          n_jobs=1), 0.68999999999999995: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
          learning_rate=1.0, n_estimators=50, random_state=0)}
Vencerdor:
OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
     verbose=0),
          n_jobs=1)
Taxa de acerto do vencedor entre os dois algoritmos no mundo real: 88.8888888889
Taxa de acerto base: 44.444444
Total de teste: 9
```

Independemente da quantidade de vezes que rodarmos o nosso algoritmo, sempre teremos o mesmo resultado para o algoritmo `AdaBoostClassifier` , portanto, o seu resultado é de 42,33%. O Nosso código final fica da seguinte forma:

```
#!-*- coding: utf8 -*-

import pandas as pd
from collections import Counter
import numpy as np
from sklearn.cross_validation import cross_val_score

texto1 = "Se eu comprar cinco anos antecipados, eu ganho algum desconto?"
texto2 = "O exercício 15 do curso de Java 1 está com a resposta errada. Pode conferir pf?"
texto3 = "Existe algum curso para cuidar do marketing da minha empresa?"

classificacoes = pd.read_csv('emails.csv')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.lower().str.split(' ')
dicionario = set()
```

```
for lista in textosQuebrados:
    dicionario.update(lista)

totalDePalavras = len(dicionario)
tuplas = zip(dicionario, xrange(totalDePalavras))
tradutor = {palavra:indice for palavra, indice in tuplas}
print totalDePalavras

def vetorizar_texto(texto, tradutor):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        if palavra in tradutor:
            posicao = tradutor[palavra]
            vetor[posicao] += 1

    return vetor

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']

X = np.array(vetoresDeTexto)
Y = np.array(marcas.tolist())


porcentagem_de_treino = 0.8

tamanho_de_treino = int(porcentagem_de_treino * len(Y))
tamanho_de_validacao = len(Y) - tamanho_de_treino

print tamanho_de_treino

treino_dados = X[0:tamanho_de_treino]
treino_marcacoes = Y[0:tamanho_de_treino]

validacao_dados = X[tamanho_de_treino:]
validacao_marcacoes = Y[tamanho_de_treino:]

def fit_and_predict(nome, modelo, treino_dados, treino_marcacoes):
    k = 10
    scores = cross_val_score(modelo, treino_dados, treino_marcacoes, cv = k)
    taxa_de_acerto = np.mean(scores)
    msg = "Taxa de acerto do {0}: {1}".format(nome, taxa_de_acerto)
    print msg
    return taxa_de_acerto

def teste_real(modelo, validacao_dados, validacao_marcacoes):
    resultado = modelo.predict(validacao_dados)

    acertos = resultado == validacao_marcacoes

    total_de_acertos = sum(acertos)
    total_de_elementos = len(validacao_marcacoes)

    taxa_de_acerto = 100.0 * total_de_acertos / total_de_elementos

    msg = "Taxa de acerto do vencedor entre os dois algoritmos no mundo real: {0}".format(taxa_de_ac
    print(msg)

resultados = {}
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
modeloOneVsRest = OneVsRestClassifier(LinearSVC(random_state = 0))
resultadoOneVsRest = fit_and_predict("OneVsRest", modeloOneVsRest, treino_dados, treino_marcacoes)
resultados[resultadoOneVsRest] = modeloOneVsRest

from sklearn.multiclass import OneVsOneClassifier
modeloOneVsOne = OneVsOneClassifier(LinearSVC(random_state = 0))
resultadoOneVsOne = fit_and_predict("OneVsOne", modeloOneVsOne, treino_dados, treino_marcacoes)
resultados[resultadoOneVsOne] = modeloOneVsOne

from sklearn.naive_bayes import MultinomialNB
modeloMultinomial = MultinomialNB()
resultadoMultinomial = fit_and_predict("MultinomialNB", modeloMultinomial, treino_dados, treino_marc
resultados[resultadoMultinomial] = modeloMultinomial

from sklearn.ensemble import AdaBoostClassifier
modeloAdaBoost = AdaBoostClassifier(random_state=0)
resultadoAdaBoost = fit_and_predict("AdaBoostClassifier", modeloAdaBoost, treino_dados, treino_marca
resultados[resultadoAdaBoost] = modeloAdaBoost

print resultados

maximo = max(resultados)
vencedor = resultados[maximo]

print "Vencerdor: "
print vencedor

vencedor.fit(treino_dados, treino_marcacoes)

teste_real(vencedor, validacao_dados, validacao_marcacoes)

acerto_base = max(Counter(validacao_marcacoes).itervalues())
taxa_de_acerto_base = 100.0 * acerto_base / len(validacao_marcacoes)
print("Taxa de acerto base: %f" % taxa_de_acerto_base)

total_de_elementos = len(validacao_dados)
print("Total de teste: %d" % total_de_elementos)
```

Resumindo

Nesse capítulo, aprendemos como podemos classificar textos utilizando os nossos algoritmos de classificação. Nosso primeiro passo armazenar todas as palavras distintas dentro de uma lista ou estrutura de dados. Entretanto para garantir que não existirão valores iguais, ou seja, se adicionarmos a palavra "se" nessa estrutura de dados, ela não poderá ser adicionada novamente, aprendemos que precisamos utilizar o mesmo conceito da teoria dos conjuntos na matemática e, no nosso código, utilizamos o `set` que tem o mesmo conceito. Além disso, vimos que precisamos transformar todas as palavras em números, isto é, para cada palavra contida no nosso texto, precisamos associá-la a um número, por exemplo, a palavra "se" será o número 0, a palavra "come" será 1 e assim por diante. Para esse problema, utilizamos a função `zip` que nos devolve um array com tuplas que são justamente pares ordenados que associam um valor ao outro, por exemplo: {se, 1}, {como, 2} e assim sucessivamente. A partir das tuplas, criamos uma estrutura que, a partir de uma palavra, permitia encontrarmos o seu número, chamamos essa estrutura de tradutor. Em seguida , criamos uma função que, dado um texto qualquer, "vetorizava-o", isto é, criava um array com a quantidade de posições igual à quantidade de palavras contidas no tradutor, e então,

consultávamos o número da palavra pelo tradutor e, a partir do número obtido, somávamos 1 no array para representar a quantidade de vezes que aquela palavra repetia no texto. Por fim, pegamos todos os arrays obtidos dessa função, extraímos todos os nossos dados e marcações a partir deles e testamos nos nossos algoritmos.