07

Quebrando na pontuação adequada

Quebrando na pontuação adequada

Conseguimos criar um algoritmo capaz de classificar os nossos e-mails em 3 categorias distintas. Esses e-mails são dados reais que eu extraí do meu dia a dia e as categorias são:

- **categoria 1**: Comercial.
- **categoria 2**: Técnico.
- **categoria 3**: Carreira.

O nosso próximo passo é justamente entender como o nosso algoritmo funcionou de acordo com o nosso dicionário, isto é, entender o processo por de trás dos panos, como por exemplo, verificar quais foram as palavras que ele utilizou. Vamos rodar o nosso arquivo `classificando_emails.py` e analisar o resultado que ele nos apresenta:

```
> 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
```

Logo no começo da impressão, ele nos informa que foram utilizadas 365 palavras, porém, quais palavras foram essas? Vamos verificar imprimindo o nosso dicionario logo após adicionarmos todas as palavras nele:

```
# restante do código

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

```
print dicionario
```

Rodando novamente o nosso arquivo `classificando_emails.py`:

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

# restante da impressão
```

Observe que dentre essas palavras, existem diversas palavras que fazem todo o sentido para analisarmos o conteúdo de um e-mail, como por exemplo: "recomendam", "carreira", "preço", "certificado", "trocar", "ferramenta" entre outras palavras que nos permite compreender a que se referem. Entretanto, ao mesmo tempo que temos palavras com um grande significado para o nosso negócio, temos também as palavras que não fazem sentido, como por exemplo: "com", "o", "uma", "isto" entre outras palavras que consideramos na língua portuguesa, isto é, utilizamos para construir os textos, mas, que não servem como parâmetro para que possamos analisar o conteúdo dos nossos e-mails. Essas palavras que não precisamos utilizar para avaliarmos o nosso texto, são chamadas de palavras de parada, ou tecnicamente, *stop words* (https://en.wikipedia.org/wiki/Stop_words).

Repare que não faz sentido termos essas stop words dentro do nosso dicionário, pois se analisarmos esse tipo de palavra, provavelmente estaremos enganando o nosso algoritmo, pois, por motivos de coincidência, ele pode tomar uma decisão que na verdade não faz muito sentido, pois ele se baseou nas stop words. Em outras palavras, quanto mais focarmos em palavras que fazem sentido para o nosso negócio, ou seja, que mostrem algum sentimento ou intenção, provavelmente o nosso algoritmo obterá melhores resultados no processo de classificação.

Um outro detalhe muito importante, é a questão do desempenho do nosso algoritmo, pois quanto mais dados utilizarmos, mais lento o algoritmo ficará, por exemplo, suponhamos que sejam 1 milhão de palavras para analisarmos, quanto tempo será que o nosso algoritmo levaria para processar? Provavelmente bem mais do que com 100 palavras ou menos. Portanto, ao realizarmos um filtro de stop worlds, além de melhorar a precisão do nosso algoritmo, também melhoramos o seu desempenho.

Porém como faríamos isso no nosso código? Leríamos todos os nossos e-mails, anotaríamos todas as palavras que consideramos como stop words, e então, pediríamos para o nosso dicionário remover uma a uma? Parece um tanto trabalhoso... Felizmente alguém já fez isso pra gente, ou seja, utilizaremos uma biblioteca que fará esse filtro pra nós! Então vamos começar criando um novo arquivo chamado `classificando_emails_limpos.py` dentro do diretório onde estão os nossos arquivos pythons. Em seguida, copie todo o código contido no arquivo `classificando_emails.py` e cole nesse arquivo que acabamos de criar. Segue abaixo o conteúdo inicial do arquivo `classificando_emails_limpos.py`:

```
#!-*- 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)

print dicionario

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)
```

Faremos essa abordagem para analisar os nossos 2 algoritmos, ou seja, verificar as alterações que fizemos no nosso código original após a limpeza. Tendo conhecimento do que será realizado daqui pra frente, podemos iniciar a nossa analise do código. Vejamos um trecho antes de imprimir o nosso dicionário:

```
dicionario = set()

for lista in textosQuebrados:
```

```
        dicionario.update(lista)

    print dicionario
```

Repare que nesse instante, estamos adicionando palavra por palavra dentro da variável `dicionario`, porém, não impomos nenhuma condição para que a palavra seja adicionada. Então o que devemos fazer? Precisamos adicionar uma condição que diz:

- Adicione essa palavras **apenas** se não fizer parte do conjunto de palavras de paradas, ou seja, stop words.

Para realizarmos essa condicação, primeiramente precisamos do conjunto de stop words. Porém, atualmente não temos esse conjunto de stop words, ou seja, faremos uso de uma biblioteca de processamento de linguagem do python, nesse caso, usaremos o National Language Tookit (http://www.nltk.org/). Porém, precisamos instalá-lo. Podemos utilizar o `pip` para isso:

```
> pip install nltk
```

Lembre-se que se você estiver utilizando Linux, precisará da permissão de super usuário, ou seja, o `sudo`.

Com o nltk instalado, podemos testá-lo dentro do interpretador do python:

```
> python
>>> import nltk
>>>
```

Qual é o nosso próximo passo? Precisamos pedir ao nltk, a partir da sua biblioteca:

```
>>> nltk.corpus
```

Todas as palavras de paradas, ou seja, as stop words:

```
>>> nltk.corpus.stopwords
```

Das palavras da língua portuguesa:

```
>>> nltk.corpus.stopwords.words("portugues")
```

Vejamos o resultado que ele nos apresenta:

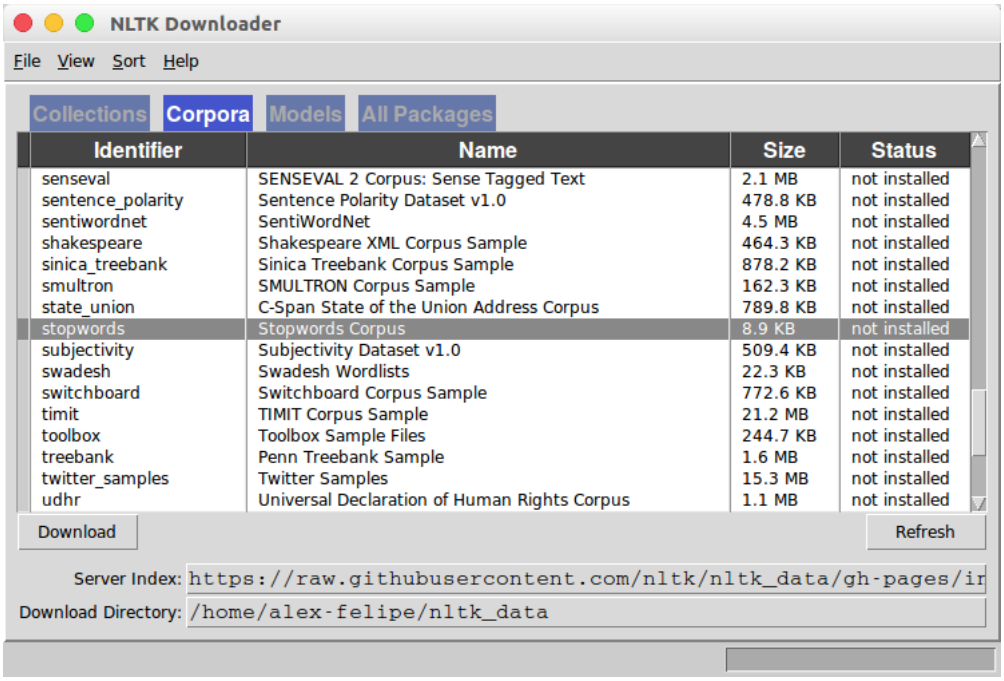
```
>>> nltk.corpus.stopwords.words("portuguese")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/nltk/corpus/util.py", line 99, in __getattr__
    self.__load()
  File "/usr/local/lib/python2.7/dist-packages/nltk/corpus/util.py", line 64, in __load
    except LookupError: raise e
LookupError:
```

```
**********************************************************************
  Resource u'corpora/stopwords' not found.  Please use the NLTK
  Downloader to obtain the resource:  >>> nltk.download()
  Searched in:
    - '/home/alex-felipe/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
**********************************************************************
>>>
```

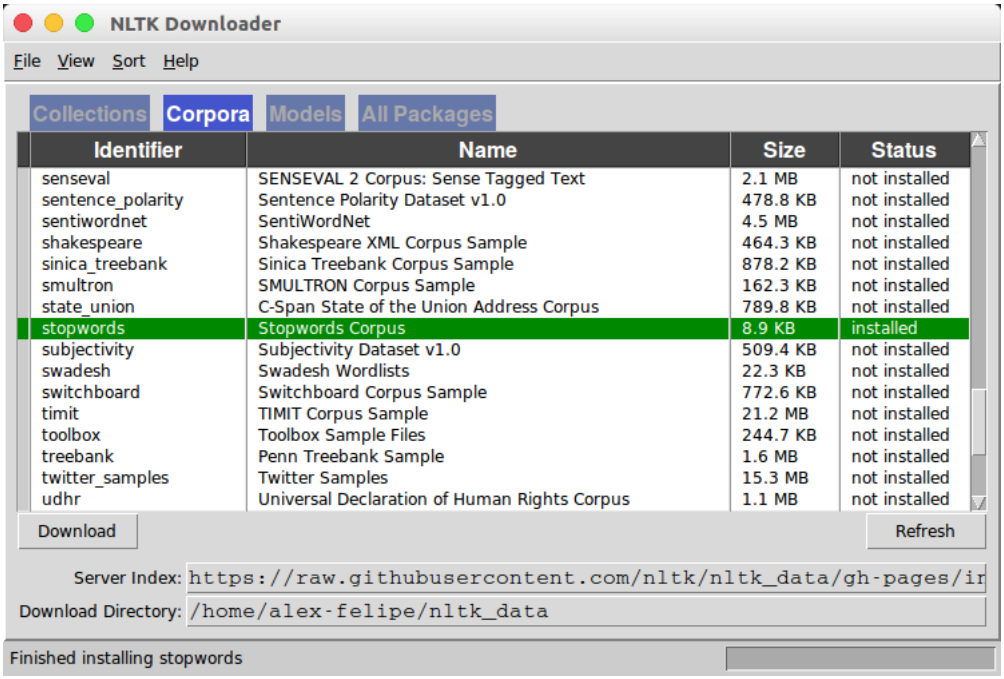
Repare que ele nos apresenta esse erro. Por que será que isso aconteceu? Será que não foi instalado corretamente? Instalamos normalmente a biblioteca do nltk, porém, essa biblioteca da suporte a diversas linguagens como por exemplo, inglês, japonês, italiano, espanhol entre outras. Lembra que a instalação do nltk foi bem rápida? Ou seja, será que no momento em que foi realizada a instalação, foram instalados também todos os pacotes que da suporte para todas as linguagens? Nesse caso não! Pois essa instalação é apenas da biblioteca, em outras palavras, as demais bibliotecas que temos interesse, como por exemplo, a de suporte ao português, precisará ser instalada a parte. Podemos ver que a própria mensagem nos diz isso, informando que o 'corpora/stopwords' não foi encontrado: *'corpora/stopwords' not found*. Em seguida ele pede para baixá-lo utilizando a instrução `nltk.download()` . Então faremos isso:

```
>>> nltk.download()
```

Após executar essa instrução, abrirá a seguinte janela:



Dentro dessa janela vá na aba "Corpora" e procure o pacote "stopwords" conforme a figura acima. Após selecionar o pacote, clique em "Download". Ao finalizar o download, será apresentada a mensagem "installed" indicando que foi instalado, conforme a figura abaixo:



Agora podemos fechar a janela para download de pacotes e voltarmos ao terminal.

```
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
True
>>>
```

Vamos tentar novamente executar o comando que pega todas as stop words da língua portuguesa:

```
>>> nltk.corpus.stopwords.words("portuguese")
[u'de', u'a', u'o', u'que', u'e', u'do', u'da', u'em', u'um', u'para', u'com', u'n\xe3o', u'uma', u'
```

O nltk nos apresenta todas as stop words da língua portuguesa, isto é, as palavras que não possuem intenções ou sentimento para a nossa regra de negócio. Consegue observar o que essas palavras tem em comum? Todas essas palavras aparecerem com muita frequência nos textos da língua portuguesa e é justamente por isso que queremos tirá-las durante a analise dos nossos textos. Basicamente, é essa a definição das stop words, por isso não adicionaremos essas palavras dentro do nosso dicionário. Então como faremos isso no nosso código? Vá até o arquivo `classificando_emails_limpos.py` , antes de criarmos o nosso dicionário, importaremos o nltk e suas stop words da língua portuguesa:

```
# restante do código

import nltk
stopwords = nltk.corpus.stopwords.words('portuguese')

dicionario = set()

for lista in textosQuebrados:
    dicionario.update(lista)

print dicionario
```

Qual é o nosso próximo passo? Ao invés de adicionarmos toda a lista dentro do dicionário, queremos colocar apenas as palavras válidas, mas quais são as palavras válidas? Inicialmente são todas as palavras contidas na variável `lista` :

```
# restante do código

import nltk
stopwords = nltk.corpus.stopwords.words('portuguese')

dicionario = set()

for lista in textosQuebrados:
  validas = lista
    dicionario.update(validas)

print dicionario
```

Por enquanto não teve nenhuma diferença. Então vamos pegar cada palavra contida na variável `lista` :

```
# restante do código

import nltk
stopwords = nltk.corpus.stopwords.words('portuguese')

dicionario = set()

for lista in textosQuebrados:
  validas = [palavra for palavra in lista]
    dicionario.update(validas)

print dicionario
```

Repara que estamos pegando cada palavra da lista, porém, ainda não estamos adicionando alguma condição para filtrá-las, portanto, precisamos adicioná-la. Em outras palavras, podemos adicionar uma palavra como válida, apenas se ele não fazer parte do conjunto das stop words:

```
# restante do código

import nltk
stopwords = nltk.corpus.stopwords.words('portuguese')

dicionario = set()

for lista in textosQuebrados:
    validas = [palavra for palavra in lista if palavra not in stopwords]
    dicionario.update(validas)

print dicionario
```

Agora estamos adicionando apenas as palavras que não fazem parte das stop words do nltk.

Anteriormente, nosso dicionário continha 365 palavras, vamos verificar quantas palavra ele conterá sem as stop words. Primeiro saia do interpretador do python e execute o arquivo `classificando_emails_limpos.py` :

```
> python classificando_emails_limpos.py
classificando_emails_limpos.py:22: UnicodeWarning: Unicode equal comparison failed to convert both a
  validas = [palavra for palavra in lista if palavra not in stopwords]
set(['', 'letra', 'email,', 'v\xc3\xaddeo', 'certificado,', 'onde', 'digita\xc3\xa7\xc3\xa3o', 's\xc

322
34
Taxa de acerto do OneVsRest: 0.656666666667
Taxa de acerto do OneVsOne: 0.723333333333
Taxa de acerto do MultinomialNB: 0.715
Taxa de acerto do AdaBoostClassifier: 0.465
{0.65666666666666662: OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, f
     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: OneVsOneClassifier(estimator=LinearSVC(C=1.0, class_weight
     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), 0.71499999999999997: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
          learning_rate=1.0, n_estimators=50, random_state=0)}
Vencerdor:
OneVsOneClassifier(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: 77.7777777778
Taxa de acerto base: 44.444444
Total de teste: 9
```

Repara que agora ele imprimiu 322 palavras, ou seja, 43 palavras a menos que o nosso dicionário anterior continha. Lembrando que todas essas 43 palavras, são as stop words, isto é, palavras que em geral não dizem sobre o que realmente o texto se trata, como por exemplo, intenções ou sentimentos.

Conseguimos limpar o nosso dicionário, ou melhor, agora, o nosso dicionário possue diversas palavras valiosas para o nosso negócio. Entretanto, ainda existe um detalhe nas nossas palavras, pois, atualmente, temos as seguintes palavras "distintas" no nosso dicionário: "deve" e "devem". Essas duas palavras são realmente tão distintas assim? Além dessas palavras, temos outros exemplos como "curso" e "cursos". Realmente faz tanta diferença essas duas palavras? Percebe que não faz muito sentido tratarmos palavras derivadas de outra também? Ou seja, palavras conjugadas no plural ou passado ou qualquer tipo de variação de uma palavra raiz, isto é, a palavra que lhe deu origem. Portanto, tanto a palavra "deve" quanto "devem" ou "deverão" ou "deveram" ou "deveria", possuem de fato o mesmo significado!

Um outro exemplo seria a diferença de gêneros, por exemplo, a palavra "aluno" e "aluna" ou então "alunos" e "alunas" são originadas da mesma palavra, ou seja, não faz sentido tratá-las como palavras distintas! Mas será que, para esse caso, teremos que resolver na mão? Felizmente, o nltk, também nos fornece uma biblioteca, uma ferramenta, que extrai a raiz de uma palavra. No nltk, temos o *stemmer* que é responsável em retirar a palavra raiz de acordo com suas variantes. O nome da biblioteca para o stemmer é `RSLPStemmer` e podemos criá-lo a partir da instrução `nltk.stem.RSLPStemmer()`. Porém, quando tentamos executá-lo pela primeira vez:

```
> python
>>> import
>>> stemmer = nltk.stem.RSLPStemmer
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 58, in __init__
    self._model.append( self.read_rule("step0.pt") )
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 67, in read_rule
    rules = load('nltk:stemmers/rslp/' + filename, format='raw').decode("utf8")
  File "/usr/local/lib/python2.7/dist-packages/nltk/data.py", line 801, in load
    opened_resource = _open(resource_url)
  File "/usr/local/lib/python2.7/dist-packages/nltk/data.py", line 919, in _open
    return find(path_, path + ['']).open()
  File "/usr/local/lib/python2.7/dist-packages/nltk/data.py", line 641, in find
    raise LookupError(resource_not_found)
LookupError:
**********************************************************************
  Resource u'stemmers/rslp/step0.pt' not found.  Please use the
  NLTK Downloader to obtain the resource:  >>> nltk.download()
  Searched in:
    - '/home/alex-felipe/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
    - u''
**********************************************************************
>>>
```

Novamente aparece aquela mensagem informando que não temos o pacote para essa biblioteca, ou seja, precisamos baixá-la. Da mesma forma como fizemos anteriormente, utilizaremos a instrução `nltk.download()`, porém, dessa vez, utilizaremos o parâmetro 'rslp', que já baixa o pacote desejado sem a necessidade de procurar no assistente de download:

```
>>> nltk.download('rslp')
[nltk_data] Downloading package rslp to /home/alex-felipe/nltk_data...
[nltk_data]   Unzipping stemmers/rslp.zip.
True
>>>
```

Agora podemos criar o nosso stemmer:

```
>>> stemmer = nltk.stem.RSLPStemmer()
>>>
```

Faremos o nosso primeiro teste, vamos pedir a raiz da palavra "amigos":

```
>>> stemmer = nltk.stem.RSLPStemmer()
>>> stemmer.stem("amigos")
u'amig'
>>>
```

amig? Vejamos para a palavra "amigas":

```
>>> stemmer = nltk.stem.RSLPStemmer()
>>> stemmer.stem("amigos")
u'amig'
>>> stemmer.stem("amigas")
u'amig'
>>>
```

Observe que a tanto a palavra amigo(a) ou amigos(as) são palavras compostas da palavra "amig", por isso ele nos retornar essa palavra. Vamos fazer mais um teste, agora com a palavra "carreira":

```
>>> stemmer.stem("carreira")
u'carr'
>>>
```

Vejamos agora no plural:

```
>>> stemmer.stem("carreira")
u'carr'
>>> stemmer.stem("carreiras")
u'carr'
>>>
```

Funcionando conforme o esperado, em outras palavras, o stemmer retorna uma raiz baseado na palavra que informamos. Entretanto, existe um detalhe importante, em qual momento informamos que queremos que ele extraia a raiz de uma palavra da língua portuguesa? Em nenhum certo? Então será que o suporte desse *stemmer* será para a língua portuguesa ou para a inglesa ou qualquer outra língua por padrão? Diferentemente do conjunto de stop words que contém diversas palavras para diferentes línguas, o `RSLPStemmer` é um extrator de raiz das palavras baseado no Removedor de Sufixo da Língua Portuguesa, portanto, essa biblioteca foi desenvolvida para a língua portuguesa. É exatamente por esse motivo que não precisamos informar qual língua estamos utilizando. Então vamos adicioná-lo no nosso código, porém, em qual parte do nosso código? Vejamos o momento em que extraímos a palavra da lista de palavras:

```
validas = [palavra for palavra in lista if palavra not in stopwords]
```

Observe que nesse instante estamos pegando uma palavra da lista de palavras que não estão contidas nas stop words. Então o que está faltando agora? No momento em que devolvemos a palavra, precisamos dizer que devolveremos o `stem` da palavra, ou seja, a raiz da palavra:

```
validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
```

Porém, antes de testarmos, precisamos declarar o stemmer no nosso código:

```
import nltk
stopwords = nltk.corpus.stopwords.words('portuguese')

stemmer = nltk.stem.RSLPStemmer()

dicionario = set()
```

```
for lista in textosQuebrados:
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
    dicionario.update(validas)

print dicionario
```

Antes de testar, vamos organizar o nosso código. Veja que temos o import do nltk no meio do nosso código, ou seja, vamos adicioná-lo na parte superior junto aos demais imports:

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

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

# restante do código
```

Agora vamos testar o nosso código e verificar o resultado:

```
> python classificando_emails_limpos.py
classificando_emails_limpos.py:25: UnicodeWarning: Unicode equal comparison failed to convert both a
  validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py:133: UnicodeWarning: Unicode equal comparis
  if word[-suffix_length:] == rule[0]:       # if suffix matches
/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py:135: UnicodeWarning: Unicode equal comparis
  if word not in rule[3]:                # if not an exception
Traceback (most recent call last):
  File "classificando_emails_limpos.py", line 25, in <module>
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 125, in stem
    word = self.apply_rule(word, 6)
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 136, in apply_rule
    word = word[:-suffix_length] + rule[2]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 5: ordinal not in range(128)
```

Repare que ele apresentou um erro na linha 25 informando o seguinte erro: "UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interpreting them as being unequal". Vejamos o código nessa linha:

```
validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
```

É justamente no momento em que estamos atribuindo as palavras válidas. A única diferença que temos em relação ao código anterior, é que adicionamos o `stemmer`. Porém, anteriormente estava funcionando sem nenhum problema. Em outras palavras, se retirarmos a instrução do `stemmer` e testarmos novamente:

```
validas = [palavra for palavra in lista if palavra not in stopwords]
```

Temos o seguinte resultado:

```
> python classificando_emails_limpos.py
classificando_emails_limpos.py:25: UnicodeWarning: Unicode equal comparison failed to convert both a
  validas = [palavra for palavra in lista if palavra not in stopwords]
set(['', 'letra', 'email,', 'v\xc3\xaddeo', 'certificado,', 'onde', 'digita\xc3\xa7\xc3\xa3o', 's\xc

322
34
Taxa de acerto do OneVsRest: 0.656666666667
Taxa de acerto do OneVsOne: 0.723333333333
Taxa de acerto do MultinomialNB: 0.715
Taxa de acerto do AdaBoostClassifier: 0.465
{0.65666666666666662: OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, f
     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: OneVsOneClassifier(estimator=LinearSVC(C=1.0, class_weight
     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), 0.71499999999999997: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
          learning_rate=1.0, n_estimators=50, random_state=0)}
Vencerdor:
OneVsOneClassifier(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: 77.7777777778
Taxa de acerto base: 44.444444
Total de teste: 9
```

O que será que aconteceu? Vejamos novamente o retorno do `stemmer` no interpretador do python a partir da palavra "carreiras", por exemplo:

```
>>> import nltk
>>> stemmer = nltk.stem.RSLPStemmer()
>>> stemmer.stem("carreiras")
u'carr'
>>>
```

Note que ele transformou a palavra em "carr", mas, perceba que ele adiciona um "u" no começo, o que isso significa? Ele é o unicode, ou seja, essa palavra está sendo trabalhada com um unicode de um unicode. Lembra que o erro que ele nos apresentou falava sobre um problemas de unicode, vejamos novamente a mensagem:

- UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interpreting them as being unequal

Na mensagem diz que ele está tentando fazer uma comparação de unicode, porém, ocorreu uma falha na conversão entre os 2 argumentos para unicode. Em outras palavras, alguma coisa não deu muito certo durante a comparação, por exemplo, ele deve ter tentado comparar um dado que não era unicode com outro que era e então não conseguiu compará-los. Para resolvermos esse problema, vamos começar pela leitura do arquivo que contém os nossos textos. Lembra que quando lemos esse arquivo, as palavras apareceram de uma forma estranha? É justamente porque não informamos ao python qual encoding utilizar, portanto, iremos informar que queremos que ele faça a leitura utilizando o encoding UTF-8:

```
classificacoes = pd.read_csv('emails.csv', encoding = 'utf-8')
```

Agora definimos qual encoding ele utilizará no momento em que ele faz a leitura do arquivo. Mas, antes de rodar o nosso código novamente, lembre-se de pegar o stem de cada palavra como fizemos anteriormente. Vamos, novamente, testar o nosso arquivo:

```
> python classificando_emails_limpos.py
Traceback (most recent call last):
  File "classificando_emails_limpos.py", line 25, in <module>
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 103, in stem
    if word[-1] == "s":
IndexError: string index out of range
```

Repare que agora ele nos apresenta um novo erro, o que será que aconteceu dessa vez? Observe que o erro refere-se a uma posição do array que não foi possível acessar, mas que posição é essa? Veja que ele nos mostra o seguinte trecho: "if word[-1] == "s":". Nesse trecho, ele tentar pegar a posição -1, isso significa que ele tentou pegar a última posição de uma palavra, porém não deu certo, mas todas as palavras possuem a última posição, certo? Se pegarmos o nosso dicionário:

```
(['', 'letra', 'email,', 'v\xc3\xaddeo', 'certificado,', 'onde', 'digita\xc3\xa7\xc3\xa3o', 's\xc3\
```

Todas as palavras possuem a última posição. Entretanto, a primeira palavra é uma palavra vazia! Em outras palavras, no momento em que ele pega a palavra vazia e tenta pegar a última posição, isto é, o último caracter da palavra, será apresentado esse erro. Como podemos resolver esse detalhe? No momento em que extraímos as palavras da lista:

```
validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords]
```

Podemos também dizer que queremos apenas as palavras com tamanho maior que 0:

```
validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords and len(palavra) :
```

Ao rodarmos novamente o nosso arquivo `classificando_emails_limpos.py` :

```
> python classificando_emails_limpos.py
set([u'almej', u'encontr', u'voc\xeas.', u'certificado,', u'melhor', u'troc', u'seguint', u'sent', u
285
34
Taxa de acerto do OneVsRest: 0.408333333333
Taxa de acerto do OneVsOne: 0.375
Taxa de acerto do MultinomialNB: 0.4
Taxa de acerto do AdaBoostClassifier: 0.406666666667
{0.40833333333333333: OneVsRestClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=True, i
     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), 0.375: OneVsOneClassifier(estimator=LinearSVC(C=1.0, class_weight=None, dual=Tr
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=0, tol=0.0001
```

```
      multi_class='ovr', penalty='l2', random_state=0, tol=0.0001,
      verbose=0),
           n_jobs=1), 0.40000000000000002: 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: 66.6666666667
Taxa de acerto base: 44.444444
Total de teste: 9
```

O nosso algoritmo funcionou normalmente. Além disso, repara que, ao invés 365 ou 322 palavras, o nosso dicionário, nesse exato momento, está contém 285 palavras, ou melhor, 285 raízes de palavras distintas. Também, não temos mais a palavra vazia e nem as stop words. Será que o nosso algoritmo já está 100% ou ainda existe um detalhe que precisamos verificar? Vamos verificar quantas palavras estão sendo computadas no primeiro vetor de palavras:

```
# 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']

print vetoresDeTexto[0]
```

Lembra qual é o primeiro texto? Vamos verificar:

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

Rodando o nosso algoritmo temos o seguinte vetor para esse texto:

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

Observe que apenas 3 palavras foram computadas, mas será que temos apenas 3 palavras dentro dessa frase? Consultando no nosso dicionário, temos os seguintes resultados para cada uma da palavras:

- **comprar**: u'compr'
- **cinco**: u'cinc'
- **anos?**: u'anos?'
- **antecipados,**: u'antecipados,'

- **ganho**: u'ganh'
- **algum**: u'algum'
- **desconto**: u'desconto?'

Veja que temos pelo menos 7 palavras contidas no nosso dicionário, porém, apenas 3 delas estão sendo computadas no momento em que a vetorizamos, isto é, pegamos palavra por palavra de um texto e convertemos para um array numérico que computa a quantidade de repetições. O que será que aconteceu? Provavelmente esse problema deve-se ao fato de estarmos adicionando a raiz de uma palavra no nosso dicionário. Mas o quão impactante isso pode ser? Se analisarmos os resultados de cada algoritmo nesse último teste que fizemos, a taxa de acerto de ambos, foram reduzidas, portanto, o simples fato de vetorizar um texto sem computar todas as suas palavras que fazem sentido, isto é, sem as stop words, piora e muito o nosso algoritmo.

Nesse instante, precisamos verificar em qual ponto do código precisamos realizar uma analise para entender o motivo pelo qual aconteceu esse problema. Começaremos analisando a partir do trecho do código em que criamos o dicionário:

```
# restante do código

dicionario = set()

for lista in textosQuebrados:
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords and len(palavr
    dicionario.update(validas)

print dicionario
```

Até esse ponto do código, realizamos os ajustes necessários e verificamos que não tem problema algum. Então vamos para o próximo:

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

Nesse trecho também não há problema pois é simplesmente pegar todas as palavras e transformar em um tradutor, ou seja, associar um número distinto para cada uma das palavras. Vejamos o próximo trecho:

```
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
```

Nessa função realizamos de fato o processo de vetorizar um texto. Em outras palavras, provavelmente existe algum detalhe dentro dessa função que precisamos ajustar. Vamos analisar passo a passo o que está acontecendo:

```
vetor = [0] * len(tradutor)
```

Nesse trecho não há problema algum, pois só estamos pegando o total de palavras que o tradutor contém para que possamos criar o vetor. Vejamos a próxima linha:

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

Observe que estamos criando um laço para iterar sobre todas as palavras contidas no texto que enviamos, por enquanto, não tem problema algum. Entretanto, logo abaixo, verificamos se a palavra contida no texto, está contida também no tradutor, porém, atualmente, o nosso tradutor contém apenas a raiz da palavra, ou seja, faz sentido verificar se existe uma determinada palavra em um tradutor que contém apenas as raízes de todas as palavras que conhecemos? Por exemplo, no texto, temos a palavra "cinco", porém, no nosso tradutor, para a palavra "cinco", temos a raiz "cinc", então vem a questão:

- "cinco" é a mesma coisa que "cinc"?

De fato são distintas! Em outras palavras, quando realizamos uma comparação entre duas strings, ambas precisam ser **estritamente** iguais! Nesse caso, temos um tradutor com as raízes das palavras, portanto, quando tivermos que verificar se uma palavra existe no tradutor, primeiro precisamos transformá-la em sua raiz para depois compará-la! Então vamos realizar essas modificações no código. Antes de adicionarmos esses detalhes que vimos, precisamos de um stemmer na função `vetorizar_texto` , portanto, adicione o stemmer como parâmetro:

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

Qual é o nosso próximo passo? É justamente pegar o stem da palavra para verificar se existe no tradutor:

```
def vetorizar_texto(texto, tradutor, stemmer):
    vetor = [0] * len(tradutor)
  for palavra in texto:
          raiz = stemmer.stem(palavra)
          if raiz in tradutor:
```

Será que não precisamos mexer no restante do código? vejamos:

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

    return vetor
```

Observe que ainda estamos pedindo a palavra inteira para o tradutor sendo que agora, precisamos pedir as suas raízes. Em outras palavras, ao invés de pedir a posição a partir de uma palavra completa, precisamos pedir a sua raiz:

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

```
            raiz = stemmer.stem(palavra)
            if raiz in tradutor:
                posicao = tradutor[raiz]
                vetor[posicao] += 1

    return vetor
```

Aparentemente, fizemos todos os ajustes necessários, vamos testar novamente o nosso arquivo `classificando_emails_limpos.py` :

```
> python classificando_emails_limpos.py
set([u'almej', u'encontr', u'voc\xeas.', u'certificado,', u'melhor', u'troc', u'seguint', u'sent', u
285
Traceback (most recent call last):
  File "classificando_emails_limpos.py", line 45, in <module>
    vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
  File "classificando_emails_limpos.py", line 38, in vetorizar_texto
    raiz = stemmer.stem(palavra)
  File "/usr/local/lib/python2.7/dist-packages/nltk/stem/rslp.py", line 103, in stem
    if word[-1] == "s":
IndexError: string index out of range
```

Novamente aquele problema de acessar o último caractere. O que será que aconteceu? Note que, quando pegamos uma palavra:

```
def vetorizar_texto(texto, tradutor, stemmer):
    vetor = [0] * len(tradutor)
    for palavra in texto:
        raiz = stemmer.stem(palavra)

    #restante do código
```

Não filtramos se elas são vazias ou não, portanto, precisamos adicionar mais uma condição que é justamente fazer todos os passos de pegar a raiz da palavra, sua posição e somar **somente** se a palavra contida no texto, tiver tamanho maior do que 0:

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

    return vetor
```

Testando novamente o nosso código, temos o seguinte resultado:

```
> python classificando_emails_limpos.py
set([u'almej', u'encontr', u'voc\xeas.', u'certificado,', u'melhor', u'troc', u'seguint', u'sent',
285
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
34
Taxa de acerto do OneVsRest: 0.676666666667
Taxa de acerto do OneVsOne: 0.651666666667
Taxa de acerto do MultinomialNB: 0.735
Taxa de acerto do AdaBoostClassifier: 0.551666666667
{0.55166666666666664: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=0), 0.67666666666666675: OneVsRestClassi

     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), 0.65166666666666673: OneVsOneClassifier(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), 0.73500000000000001: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)]
Vencerdor:
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Taxa de acerto do vencedor entre os dois algoritmos no mundo real: 77.7777777778
Taxa de acerto base: 44.444444
Total de teste: 9
```

Repara que agora foram contabilizadas as 7 palavras da frase que havíamos analisado! Agora estamos realizando todos os passos da forma adequada, isto é, dado um conjunto de palavras, extraímos suas raizes para diminuir uma quantidade considerável de palavras a serem analisadas, melhorando tanto a performance do nosso algoritmo como também a sua precisão, pois ele terá menos variáveis para serem analisadas como, nesse caso, stop words ou palavras compostas que não fazem tanta diferença para a tomada de decisão do nosso algoritmo.

Há mais um detalhe que precisamos analisar em nosso dicionário, pois ainda existem palavras como:

- **u'eu,'**
- **minutos.**
- **empresa?**

Percebe que cada uma dessas palavras estão concatenadas com algum tipo de pontuação? Ou seja, a palavra "eu" tem uma vírgula, a "minutos" um ponto final e a "empresa" um ponto de interrogação. O Stemmer não é capaz de retirar essas pontuações nos nossos textos, portanto, se existe palavras como "eu," ou "eu.", serão consideradas palavras distintas. Em outras palavras, em nenhum momento trabalhamos com pontuação. Então o que precisamos fazer para retirar esses pontos durante a separação das palavras? Nesse caso, precisamos verificar como estamos quebrando as palavras, ou seja, qual é o critério que estamos utilizando para separar palavra por palavra dentro de um texto. Então vejamos como estamos fazendo atualmente:

```
classificacoes = pd.read_csv('emails.csv', encoding = 'utf-8')
textosPuros = classificacoes['email']
textosQuebrados = textosPuros.str.lower().str.split(' ')

# restante do código
```

Note que estamos separando cada palavra no momento em que encontramos um espaço. Ou seja, ao invés de apenas quebrar com espaços em branco, precisamos quebrar também com a pontuação. Da mesma forma como fizemos com as stop words e as palavras raízes, utilizaremos uma biblioteca que é capaz de separar palavras de um texto, tanto pelos espaços em branco

quanto pela pontuação. Essa biblioteca se chama *tokenize* e, da mesma forma como as outras, ela faz parte do conjunto de bibliotecas do nltk e precisa ser baixada. Usaremos a instrução `nlkt.download("punkt")` que é justamente o módulo para pontuação:

```
>>> import nltk
>>> nltk.download("punkt")
[nltk_data] Downloading package punkt to /home/alex-
[nltk_data]     felipe/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
>>>
```

Antes de adicionarmos no nosso código, vamos realizar um teste. A partir dessa biblioteca, pediremos para separar palavra por palavra da frase abaixo:

- "Voce vai viajar? Este ano, eu penso que sim! Claro!?Mas e voce? Sim!".

Note que essa frase é apenas um teste, portanto, estamos utilizando espaços em branco, pontuação com espaço em branco e pontuação junto com palavras em ambos os lados. Para testar utilizaremos a seguinte instrução:

```
>>> nltk.tokenize.word_tokenize("Voce vai viajar? Este ano, eu penso que sim! Claro!?Mas e voce? Sir
['Voce', 'vai', 'viajar', '?', 'Este', 'ano', ',', 'eu', 'penso', 'que', 'sim', '!', 'Claro', '!',
>>>
```

Observe que a função `word_tokenize` foi capaz quebrar todas as palavras por meio da pontuação. Porém, diferentemente do Stemmer, o `tokenize` não é exclusivo da língua portuguesa, portanto, ele pode variar de acordo com a linguagem.

Agora, vamos adicionar o `tokenize` no nosso código. Mas, em qual momento precisamos adicionar o `tokenize` ? O nosso primeiro passo é justamente separar frase por frase, portanto vamos extrair cada frase para uma variável chamada `frases` antes mesmo de realizarmos a separação de palavras:

```
classificacoes = pd.read_csv('emails.csv', encoding = 'utf-8')
textosPuros = classificacoes['email']
frases = textosPuros.str.lower()

# restante do código
```

Em seguida, para cada frase contida na variável `frases` :

```
for frase in frases
```

Precisamos devolver a `frase` processada pelo `tokenize` :

```
nltk.tokenize.word_tokenize(frase) for frase in frases
```

Transformamos esse resultado em um array e devolvemos para a variável `textosQuebrados` :

```
classificacoes = pd.read_csv('emails.csv', encoding = 'utf-8')
textosPuros = classificacoes['email']
frases = textosPuros.str.lower()
textosQuebrados = [nltk.tokenize.word_tokenize(frase) for frase in frases]

# restante do código
```

Vamos testar o nosso código e verificar o resultado:

```
> python classificando_emails_limpos.py
set([u'almej', u'ol\xe1', u'encontr', u'distanc', u'troc', u'nenhum', u'sent', u'via', u'\xe1re', u'
244
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
34
Taxa de acerto do OneVsRest: 0.768333333333
Taxa de acerto do OneVsOne: 0.755
Taxa de acerto do MultinomialNB: 0.763333333333
Taxa de acerto do AdaBoostClassifier: 0.626666666667
{0.62666666666666671: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=0), 0.75500000000000012: OneVsOneClassifi
     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), 0.76333333333333342: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=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)}
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: 77.7777777778
Taxa de acerto base: 44.444444
Total de teste: 9
```

Repara que agora o nosso dicionário contém 244 palavras. Entratanto, quando observarmos algumas palavras temos os seguintes resultados:

- u','
- u'?'
- u'.'
- u'!'
- u':'

Faz sentido tratarmos esses pontos como uma palavra? Com certeza não! Portanto, iremos descartar também essas pontuações. Mas como podemos fazer isso?

Durante todo esse processo de separar cada palavra para adicionar ao nosso dicionário, é muito comum desconsiderarmos palavras com 2 ou menos caracteres. Em outras palavras, se uma palavra tiver apenas 2 letras ou menos, podemos descartá-

la. Dessa forma, além de eliminar palavras que não fazem tanta diferença durante a classificação, resolvemos o problemas das pontuações. Então vamos alterar no nosso código no momento em que estamos inserindo as palavras no nosso dicionário:

```
# restante do código

dicionario = set()

for lista in textosQuebrados:
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords and len(palavr
    dicionario.update(validas)

print dicionario
```

Ao invés de adicionar palavras com a quantidade de caracteres maior que 0, pediremos para adicionar palavras com pelo menos 3 caracteres, ou seja, maior do que 2:

```
dicionario = set()

for lista in textosQuebrados:
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords and len(palavr
    dicionario.update(validas)

print dicionario
```

Testando novamente o nosso arquivo `classificando_emails_limpos.py` :

```
> python classificando_emails_limpos.py
set([u'almej', u'ol\xe1', u'encontr', u'distanc', u'troc', u'nenhum', u'sent', u'via', u'\xe1re', u'
230
[0, 0, 0, 0, 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,
34
Taxa de acerto do OneVsRest: 0.801666666667
Taxa de acerto do OneVsOne: 0.801666666667
Taxa de acerto do MultinomialNB: 0.83
    Taxa de acerto do AdaBoostClassifier: 0.626666666667
{0.62666666666666671: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=0), 0.80166666666666675: OneVsOneClassifi
     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), 0.83000000000000007: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Vencerdor:
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Taxa de acerto do vencedor entre os dois algoritmos no mundo real: 77.7777777778
Taxa de acerto base: 44.444444
Total de teste: 9
```

Veja que agora não temos mais pontuações ou palavras com menos de 3 caracteres no nosso dicionrário. Além disso, agora o nosso dicionário contém 230 palavras distintas! 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
import nltk

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', encoding = 'utf-8')
textosPuros = classificacoes['email']
frases = textosPuros.str.lower()
textosQuebrados = [nltk.tokenize.word_tokenize(frase) for frase in frases]

stopwords = nltk.corpus.stopwords.words('portuguese')

stemmer = nltk.stem.RSLPStemmer()

dicionario = set()

for lista in textosQuebrados:
    validas = [stemmer.stem(palavra) for palavra in lista if palavra not in stopwords and len(palavr
    dicionario.update(validas)

print dicionario

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 len(palavra) > 0:
            raiz = stemmer.stem(palavra)
            if raiz in tradutor:
                posicao = tradutor[raiz]
                vetor[posicao] += 1

    return vetor

vetoresDeTexto = [vetorizar_texto(texto, tradutor) for texto in textosQuebrados]
marcas = classificacoes['classificacao']
print vetoresDeTexto[0]


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 o nosso foco foi justamente na limpeza dos dados que estávamos utilizando para o nosso algoritmo. Inicialmente, vimos que muitas palavras como por exemplo "com", "o", "uma", "isto" entre outras palavras que são consideradas para a construção de um texto da língua portuguesa, porém, não faz sentido considerá-las para classificação. Aprendemos que essas palavras são consideradas como stop words e, justamente por não fazerem diferença durante a nossa classificação de texto, as desconsideramos. Para isso, utilizamos o pacote de bibliotecas do nltk, nesse caso, o módulo corpus para a língua portuguesa, a partir da seguinte instrução: `nltk.corpus.stopwords.words("portuguese")`

Mesmo retirando as palavras de paradas (stop words), notamos que não era o suficiente, pois ainda existiam palavras no nosso dicionário que estavam sendo consideradas "distintas", porém que continham o mesmo significado, por exemplo:

- "trocar" ou "troca".
- "curso" ou "cursos".

Vimos que cada uma dessas palavras contém uma palavra raiz, isto é, uma palavra a qual foram originadas e, para o nosso algoritmo, o importante seria lidar apenas com as raízes das palavras, justamente para reduzir a quantidade de palavras que ele teria que lidar durante a vetorização de textos, melhorando, consideravelmente,s sua performance e resultado. Da mesma forma que fizemos com as stop words, utilizamos o stemmer, uma biblioteca do nltk, que nos permitiu adicionar apenas a raiz da palavra dentro do dicionário, reduzindo consideravelmente a quantidade de palavras que continhamos no nosso dicionário.

Por fim, vimos que mesmo realizando o filtro de stop words e a raiz das palavras, ainda continhamos palavras como:

- **u'eu,'**
- **minutos.**
- **empresa?**

Note que são palavras concatenadas com pontuações. Para essa situação, verificamos que o problema estava justamente no momento em que separamos cada palavra do texto, ou seja, utilizamos apenas o critério de espaços vazios. Resolvemos esse problema de separar as palavras tanto por espaços em brancos quanto para pontuações, utilizamos mais um módulo do nltk, para essa situação, o tokenize que separava palavras de espaços em brancos e pontuação. Mas, além disso, consideramos apenas palavras com mais de 2 caracteres, pois é bem comum desconsiderarmos essas palavras que não darão tanto significado durante à classificação e, dessa forma, também resolvemos o problema que adicionava as pontuações no nosso dicionário.