ProyectoNLP-BRC-AAIMD

December 7, 2020

[1]:
```
from math import inf
from collections import Counter
from collections import OrderedDict
```

1 1.Codigo de norving

[2]:
```
"""
    Spelling Corrector in Python 3; see http://norvig.com/spell-correct.html

    Copyright (c) 2007-2016 Peter Norvig
    MIT license: www.opensource.org/licenses/mit-license.php
"""
################ Spelling Corrector ################
######################################################
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or␣
 ↪[word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
```

```
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in␣
 ↪letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))

################ Test Code

def unit_tests():
    assert correction('speling') == 'spelling','Err: insert'# insert
    assert correction('korrectud') == 'corrected'           # replace 2
    assert correction('bycycle') == 'bicycle'               # replace
    assert correction('inconvient') == 'inconvenient'       # insert 2
    assert correction('arrainged') == 'arranged'            # delete
    assert correction('peotry') =='poetry'                  # transpose
    assert correction('peotryy') =='poetry'                 # transpose + delete
    assert correction('word') == 'word'                     # known
    assert correction('quintessential') == 'quintessential' # unknown
    assert words('This is a TEST.') == ['this', 'is', 'a', 'test']
    assert Counter(words('This is a test. 123; A TEST this is.')) == (
           Counter({'123': 1, 'a': 2, 'is': 2, 'test': 2, 'this': 2}))
    assert len(WORDS) == 32198
    assert sum(WORDS.values()) == 1115585
    assert WORDS.most_common(10) == [
        ('the', 79809),
        ('of', 40024),
        ('and', 38312),
        ('to', 28765),
        ('in', 22023),
        ('a', 21124),
        ('that', 12512),
        ('he', 12401),
        ('was', 11410),
        ('it', 10681)]
    assert WORDS['the'] == 79809
    assert P('quintessential') == 0
```

```
    assert 0.07 < P('the') < 0.08
    return 'unit_tests pass'
```

[3]:
```
print(unit_tests())
print(correction('speling'))
print(correction('korrectud'))
print(correction('thu'))
```

```
unit_tests pass
spelling
corrected
the
```

2 2.La siguiente palabra m'as probable

Usando *big.txt* crear una funci'on que estime la siguiente palabra m'as probable dada una anterior. La funci'on debbe calcular

$$w_{i+1} = \text{argmax}_{w_{i+1}} P(W_{i+1}|w_i)$$

Para este trabajo 1. Podemos asumir que ambas palabras siempre existir'an en la colecci'on 2. Requerimos una funci'on similar a P, que calcule $P(w_1|w_2)$

[4]:
```
################################
### Funciones para trabajar  ###
################################

def words_from_file( fileName ):
    """ Obtenemos las palabras de un archivo. """
    file = open(fileName).read()
    return re.findall(r'\w+', file.lower())

def create_dict(texto):
    """ Funcion para crear el diccionario auxiliar para
    calcular las probabilidades necesarias.
    """
    ret = {}
    for i in range(1,len(texto)):
        if texto[i] not in ret:
            ret[texto[i]] = {}
        if texto[i-1] not in ret[texto[i]]:
            (ret[texto[i]])[texto[i-1]] = 0

        (ret[texto[i]])[texto[i-1]] += 1

    # Pre-ordenado
    for word in ret:
        ret[word] = OrderedDict(sorted(ret[word].items(),
                                       key=lambda x:
```

```
                                        prob_cond(x[0], word, ret),
                                        reverse=True))

    return ret

def prob_cond(a, b, dic):
    """ Probabilidad de A dado B en funcion de dic """
    try:
        return ((dic[a])[b])/sum(dic[b].values())
    except KeyError:
        return -1

def next_word(word, dic):
    """ Obtenemos la siguiente palabra mas probable en funcion
    del dicionario y sus probabiliodades. """
    try:
        return next(iter(dic[word]))
    except:
        return word
```

[5]:
```
dic = create_dict(words_from_file('big.txt'))
word = 'new'

print( word +' '+next_word( word, dic) )
print( prob_cond('york','new', dic) )
```

```
new york
0.15811258278145696
```

2.1 2.1.Aqu'i la maquina juega al ahorcado

Se recomienda extender y mejorar de alg'un modo la funci'on propuesta por **Norving**.

[6]:
```
def under(word):
    word = word.split('_')
    if len(word) > 5:
        print('Demasiadas letras desconocidas')
        return None
    return word

def candidatos(word):
    '''  Recibe a word ya con el 'split' aplicado
        y regresamos las posibles palabras
    '''
    letters = 'abcdefghijklmnopqrstuvwxyz'
    n_letters = len(letters)
    flag = word[-1] if word[-1] != '' else 'BrendA'
```

```
    # Creamos los posibles 'pedacitos' de la palabra
    words = [ele + letter
             for ele in word[:len(word)-1]
             for letter in letters]

    # Variables auxiliares
    options = words[:n_letters]
    options_t = []

    # Concatenamos los posibles 'pedacitos'
    for k in range( 1, len(words)//n_letters ):
        for option in options:
            for i in range(n_letters):
                options_t.append(option + words[n_letters*k + i])
        options = options_t; options_t = []

    if flag != 'BrendA': # Checamos si al final hay un '_' o una letra
        for i in range(len(options)):
            options[i] = options[i] + flag

    # Regresamos unicamente las palabras que esten en el diccionario
    return set(opt for opt in options if opt in WORDS)

def dist_lev(source, target):
    if source == target: return 0
    # Crear matriz
    n_s, n_t = len(source), len(target)
    dist = [[0 for i in range(n_t+1)] for x in range(n_s+1)]
    for i in range(n_s+1): dist[i][0] = i
    for j in range(n_t+1): dist[0][j] = j
    # Calculando la distancia
    for i in range(n_s):
        for j in range(n_t):
            cost = 0 if source[i] == target[j] else 1
            dist[i+1][j+1] = min(
                                dist[i][j+1] + 1,   # deletion
                                dist[i+1][j] + 1,   # insertion
                                dist[i][j] + cost   # substitution
                            )
    return dist[-1][-1]

def closest(word, options):
    ret = 'BrendA', inf
    for opt in options:
        dist = dist_lev(word, opt)
        ret = (opt, dist) if dist < ret[1] else ret
    return ret
```

```
def hangman(word):
    options = candidatos( under(word) )
    return closest(word, options)
```

[7]:
```
print(hangman('s_e_l_c_')[0]) #sherlock
print(hangman('no_eb_o_')[0]) #notebook
print(hangman('he__o')[0])    #hello

print(hangman('pe_p_e')[0]) #people
print(hangman('phi__sop_y')[0]) #philospphy
print(hangman('si_nif_c_nc_')[0]) #significance
print(hangman('kn__l_d_e')[0])      #sun
```

```
sherlock
notebook
hello
people
philosophy
significance
knowledge
```

2.2 2.2.Ahorcado al extremo

Unir la funci'on de *2* y *2.1* para, utilizando una palabra de contexto, completar palabras con mayor precisi'on

[8]:
```
def super_under(word):
    ct = Counter(word)
    if len(word) - ct['_'] < 1:
        print('Demasiadas letras desconocidas')
        return None
    word = word.split('_')
    return word

def super_closest( context, options):
    ret = 'BrendA', -inf
    for opt in options:  # Buscando el ret adecuado
        # Esta es la misma funcion de probabilidad del ejercicio anterior
        prob = prob_cond(opt, context, dic)
        #  En caso de que las proabilidades empaten
        # utilizamos las distancia entre las palabras
        # para responder.
        ret = ((opt, prob) if dist_lev(context, opt) < dist_lev(context,␣
 ↪ret[0]) else ret) if prob == ret[1] else ret
        ret = (opt, prob) if prob > ret[1] else ret
    return ret
```

```
def super_hangman(context, word):
    options = candidatos( super_under(word) )
    return super_closest(context, options)
```

[9]:
```
print(super_hangman('sherlock', '_____s'))  #holmes
print(super_hangman('united', '_t_t__'))    #states
print(super_hangman('white', '___s_'))      #house
print(super_hangman('new', 'y___'))         #york
print(super_hangman('abraham', 'l_____n'))  #lincoln
```

```
('holmes', 1.0)
('states', 0.7620751341681574)
('house', 0.037142857142857144)
('york', 0.15811258278145696)
('lincoln', 0.6666666666666666)
```

3 3.Correci'on ortografica simple

3.0.1 Funciones auxiliares

[10]:
```
import os, re

# simple extraction of words
def words (text) :
    return re.findall(r'\w+', text.lower())

# siple loading of the documents
from keras.preprocessing.text import Tokenizer
def get_texts_from_catdir( catdir ):
    texts = [ ]
    TARGET_DIR = catdir # "./target"
    for f_name in sorted( os.listdir( TARGET_DIR )) :
        if f_name.endswith('.txt'):
            f_path = os.path.join( TARGET_DIR, f_name )
            #print(f_name)
            #print(f_path)
            f = open( f_path , 'r', encoding='utf8' )
            #print( f_name )
            texts += [ f.read( ) ]
            f.close( )
    print( '%d files loaded . ' %len(texts) )
    return texts

# Load the RAW text
target_txt = get_texts_from_catdir( './target' )

# Print first 10 words in document0
```

```
print( words(target_txt[0])[:10] )
```

```
10 files loaded .
['scientists', 'witness', 'huge', 'cosmic', 'crash', 'find', 'origins', 'of',
'gold', 'even']
```

3.0.2 Mezclar diccionarios

[11]:
```
import json

WORDS = Counter(words(open('big.txt').read()))
with open('WORDS_IN_NEWS.txt', 'r') as infile: # Exportando WORDS_IN_NEWS
    WORDS_IN_NEWS = json.load( infile )
WORDS_IN_NEWS = Counter(WORDS_IN_NEWS)

WORDS = WORDS + WORDS_IN_NEWS
print(WORDS['the'])
print(WORDS.most_common(5))
```

```
80337
[('the', 80337), ('of', 40265), ('and', 38564), ('to', 29063), ('in', 22262)]
```

3.0.3 Detectar las plabras mal escritas

[12]:
```
def mispelled_and_candidates( target_words ):
    mispelled_candidates = []
    for word in target_words:
        temp = list(candidates(word))  # candidates de Norving
        if len(temp) > 1:
            temp.sort(key=lambda x: dist_lev(word, x))
            mispelled_candidates.append((word, temp[:10])) #Tomamos las␣
 ↪primeras 10
    return mispelled_candidates

def mispelled_and_candidates( target_words ):
    mispelled_candidates = []

    for word in target_words:
        candidatos = list(candidates(word))
        candidatos.sort(key=lambda x: dist_lev(word, x))
        if len(candidatos) > 1:
            # En caso de que haya una opcion
            mispelled_candidates.append((word, candidatos[:10]))
        elif len(candidatos) == 1 and word not in candidatos:
            # En caso de que la unica opcion sea distinta
            mispelled_candidates.append((word, candidatos))
```

```
    return mispelled_candidates

#print ( mispelled_and_candidates( words( target_txt[0] )))

# Print misspelled words and candidates for each document in
# target_txt  list
for text in target_txt:
    print ( mispelled_and_candidates ( words ( text ) ) )
    pass
```

```
[('detcted', ['detected']), ('intoo', ['into'])]
[('conttinue', ['continue'])]
[('thhe', ['thee', 'the'])]
[('statment', ['statement'])]
[('watchng', ['watching'])]
[('possiblle', ['possible'])]
[('saiid', ['said'])]
[('addresss', ['address', 'addresses'])]
[('essetially', ['essentially'])]
[('gennerral', ['general'])]
```

3.0.4 Correccion completa

Para este ejercicio supondremos que la primera palabra esta bien escrita y tiene sentido.

La funcion `spell_correction` tiene una caracteristica que puede o no mejorar dependiendo de ciertos casos. De manera general, primero pasamos por la funcion del iniciso anterior al texto e identificamos todas las palabras mal escritas, luego, priorizando la probabilidad que ofrece la palabra anterior, escogemos la mejor opcione de entre aquellas que se generen por `candidates` *de Norvng.*

Esta forma de actuar tiene la principal desventaja de que no detectara problemas como las ultimas dos pruebas (ejemplos) que se proponen. Donde son palabras bien escritas pero que no necesariamente son las correctas, para solucionar esto podemos dar una propuesta mas agresva donde, en caso de que la palabra que probabilisticamente halbando (y en funcion con el corpus) deberia de seguir, la ponemos sin preguntar. Esto permite solucionar mas incisos del ejemplo, pero tambien descompone otras partes (como se puede ver en las pruebas de las noticias)

En general creo que aqui es donde podemos darle la opcion al humano para que escoja la palabra que mejor se acomode. Para superar esto podriamos ampliar el corpus o considerar la palabra que mejor se complemente con la que sigue. En caso de empezar con estasconsideraciones me parece que seria mejor primero arreglar todos las palabras que estan claramente mal escritas y luego hacer otra pasada con probabilidades.

Nota Dado que *ham* no parece estar en el corpus, causa problemas

[13]:
```
# Creacion de diccionario ampliado
# Aunque no sirve de mucho
nbig = open('big.txt').read()
```

```
for text in target_txt:
    nbig += text

dic = create_dict(words(nbig))
```

[14]:
```
def spell_correction( input_text, max_dist=2, profundo=False):
    """ Profundo le da mas libertad a la maquia para mejorar el texto. """
    corrected_text = input_text
    mispeled = dict(mispelled_and_candidates(input_text))

    for iw in range(1, len(input_text)):
        pword = corrected_text[iw-1]

        word = input_text[iw]
        nword = next_word(pword, dic)

        # En otro caso consideramos las probabilidades
        if word in mispeled:
            corrected_text[iw] = max(mispeled[word],
                                     key=lambda x: prob_cond(x, pword, dic))
        # Si se parecem cambiamos sin preguntar
        if profundo and dist_lev(nword, word) <= max_dist:
            corrected_text[iw] = nword

    return corrected_text

tests = [['i', 'hav', 'a', 'ham'],
     ['my', 'countr', 'is', 'biig'],
     ['i', 'want', 't00', 'eat'],
     ['the', 'science', '0ff', 'computer'],
     ['the', 'science', 'off', 'computer'],
     [ 'i', 'want' , 'too' , 'eat']
    ]
for s in tests:
    #print(mispelled_and_candidates(s))
    print(s)
    print( spell_correction( s, profundo=True ))
    print()
```

```
['i', 'hav', 'a', 'ham']
['i', 'have', 'a', 'man']

['my', 'countr', 'is', 'biig']
['my', 'country', 'is', 'big']

['i', 'want', 't00', 'eat']
['i', 'want', 'to', 'eat']
```

```
['the', 'science', '0ff', 'computer']
['the', 'science', 'of', 'computer']

['the', 'science', 'off', 'computer']
['the', 'science', 'of', 'computer']

['i', 'want', 'too', 'eat']
['i', 'want', 'to', 'eat']
```

Chequeo con Golden

[15]:
```
golden_txt = get_texts_from_catdir( './golden' )
golden_words = words(" ".join(golden_txt))
target_words = words(" ".join(target_txt))

i = 0
for gword, tword in zip(golden_words, target_words):
    if gword != tword:
        print(f"{i} => {gword} != {tword}")
        i+=1
```

```
10 files loaded .
0 => detected != detcted
1 => into != intoo
2 => continue != conttinue
3 => the != thhe
4 => statement != statment
5 => watching != watchng
6 => possible != possiblle
7 => said != saiid
8 => address != addresss
9 => essentially != essetially
10 => general != gennerral
```

[16]:
```
new_text = spell_correction(target_words)
new_words = words(" ".join(new_text))

i = 0
for gword, nword in zip(golden_words, new_words):
    if gword != nword:
        print(f"{i} => {gword} != {nword}")
        i+=1
else:
    if i==0:
        print("<-----|!!! No hay errores =D !!!|----->")
```

```
<-----|!!! No hay errores =D !!!|----->
```

[18]:
```
new_text = spell_correction(target_words, profundo=True)
new_words = words(" ".join(new_text))

i = 0
for gword, nword in zip(golden_words, new_words):
    if gword != nword:
        print(f"{i} => {gword} != {nword}")
        i+=1
else:
    if i==0:
        print("<-----|!!! No hay errores =D !!!|----->")
    else:
        print(" ='( Ahora si )'=")
```

```
0 => in != if
1 => ago != and
2 => the != that
3 => two != the
4 => to != as
5 => of != a
6 => a != man
7 => star != stars
8 => a != to
9 => a != it
10 => an != a
11 => to != in
12 => on != i
13 => the != to
14 => one != be
15 => we != the
16 => in != of
17 => would != gold
18 => s != of
19 => at != of
20 => we != to
21 => ve != be
22 => this != the
23 => the != to
24 => on != and
25 => 5 != to
26 => 88 != be
27 => the != to
28 => in != and
29 => how != to
30 => are != and
31 => for != to
```

```
32 => 4 != in
33 => the != he
34 => this != the
35 => s != of
36 => out != you
37 => the != are
38 => 1 != he
39 => are != he
40 => a != had
41 => was != is
42 => to != s
43 => of != a
44 => this != the
45 => to != of
46 => the != her
47 => that != the
48 => into != it
49 => said != and
50 => has != he
51 => 60 != in
52 => do != he
53 => that != the
54 => have != he
55 => was != is
56 => said != david
57 => in != and
58 => we != he
59 => in != and
60 => said != and
61 => the != to
62 => one != the
63 => on != and
64 => any != a
65 => that != this
66 => get != be
67 => in != and
68 => the != he
69 => said != david
70 => the != her
71 => this != the
72 => they != the
73 => 20 != in
74 => a != and
75 => to != the
76 => we != he
77 => at != a
78 => we != of
79 => this != him
```

```
80 => it != he
81 => has != had
82 => been != then
83 => a != he
84 => as != to
85 => to != it
86 => in != twin
87 => the != those
88 => on != of
89 => as != last
90 => the != to
91 => in != and
92 => other != over
93 => in != and
94 => at != of
95 => up != in
96 => and != a
97 => it != him
98 => this != the
99 => has != al
100 => to != of
101 => their != the
102 => as != last
103 => as != of
104 => man != and
105 => in != and
106 => the != them
107 => 70 != in
108 => to != as
109 => had != and
110 => he != a
111 => to != of
112 => the != to
113 => s != us
114 => a != and
115 => is != he
116 => two != the
117 => has != al
118 => a != and
119 => of != _
120 => in != of
121 => at != and
122 => a != in
123 => said != and
124 => the != them
125 => to != the
126 => we != the
127 => in != and
```

```
128 => we != of
129 => for != to
130 => a != be
131 => for != of
132 => those != the
133 => said != and
134 => face != same
135 => this != the
136 => the != her
137 => to != the
138 => at != and
139 => for != to
140 => in != and
141 => the != he
142 => the != that
143 => to != of
144 => the != there
145 => an != a
146 => to != a
147 => after != later
148 => in != and
149 => to != a
150 => for != of
151 => a != of
152 => he != a
153 => was != man
154 => a != and
155 => this != the
156 => he != a
157 => was != man
158 => by != s
159 => in != to
160 => to != in
161 => at != by
162 => bay != at
163 => a != by
164 => a != of
165 => the != he
166 => be != of
167 => to != not
168 => a != to
169 => is != he
170 => to != the
171 => a != and
172 => for != of
173 => his != him
174 => who != two
175 => in != and
```

```
176 => he != the
177 => case != same
178 => t != i
179 => as != of
180 => is != he
181 => out != not
182 => in != and
183 => to != of
184 => he != the
185 => he != it
186 => his != it
187 => s != of
188 => to != he
189 => the != be
190 => of != to
191 => he != of
192 => he != the
193 => he != him
194 => he != the
195 => said != same
196 => that != they
197 => then != this
198 => plan != man
199 => to != s
200 => to != the
201 => had != have
202 => i != he
203 => was != had
204 => to != on
205 => the != be
206 => i != he
207 => was != had
208 => i != he
209 => be != he
210 => what != had
211 => those != the
212 => go != he
213 => the != be
214 => those != the
215 => they != the
216 => that != the
217 => to != he
218 => i != to
219 => i != is
220 => me != to
221 => i != he
222 => am != had
223 => i != a
```

```
224 => don != man
225 => t != s
226 => said != had
227 => 20 != in
228 => they != the
229 => for != of
230 => an != it
231 => the != her
232 => of != it
233 => for != to
234 => we != a
235 => t != i
236 => any != away
237 => to != a
238 => he != the
239 => t != i
240 => he != it
241 => the != when
242 => s != in
243 => 7 != a
244 => was != is
245 => no != to
246 => that != the
247 => in != its
248 => that != the
249 => as != do
250 => woman != man
251 => her != be
252 => i != a
253 => i != it
254 => ve != is
255 => not != to
256 => ago != as
257 => then != the
258 => it != a
259 => me != the
260 => 34 != it
261 => an != and
262 => to != the
263 => was != is
264 => did != i
265 => the != her
266 => an != a
267 => and != a
268 => one != and
269 => in != do
270 => an != in
271 => the != her
```

```
272 => off != of
273 => on != of
274 => to != of
275 => a != be
276 => the != be
277 => to != of
278 => it != or
279 => the != one
280 => see != are
281 => a != be
282 => on != to
283 => t != be
284 => on != and
285 => it != a
286 => if != i
287 => have != are
288 => it != i
289 => they != he
290 => to != tv
291 => to != the
292 => of != not
293 => a != to
294 => set != be
295 => top != the
296 => now != for
297 => the != he
298 => at != to
299 => t != be
300 => to != the
301 => has != he
302 => a != had
303 => s != an
304 => use != the
305 => if != of
306 => re != are
307 => tie != be
308 => a != and
309 => as != and
310 => at != a
311 => in != and
312 => for != of
313 => to != you
314 => up != to
315 => tv != an
316 => in != and
317 => that != the
318 => the != then
319 => a != to
```

320 => lg != a
321 => by != to
322 => the != be
323 => the != he
324 => ends != and
325 => 50 != to
326 => per != be
327 => it != i
328 => time != the
329 => in != and
330 => at != not
331 => use != be
332 => this != the
333 => s != a
334 => use != be
335 => that != the
336 => the != to
337 => 7 != a
338 => a != man
339 => on != and
340 => at != to
341 => at != a
342 => tv != the
343 => to != the
344 => tv != so
345 => use != be
346 => that != the
347 => tv != the
348 => on != a
349 => a != man
350 => tv != of
351 => the != he
352 => or != your
353 => its != to
354 => app != and
355 => for != to
356 => a != in
357 => s != tv
358 => you != to
359 => to != be
360 => tv != or
361 => 50 != to
362 => a != be
363 => has != he
364 => an != to
365 => to != the
366 => and != in
367 => it != a

```
368 => a != to
369 => rob != for
370 => a != to
371 => d != and
372 => c != a
373 => a != to
374 => e != of
375 => at != and
376 => him != it
377 => on != is
378 => s != is
379 => got != to
380 => or != for
381 => now != not
382 => be != to
383 => able != be
384 => to != the
385 => and != a
386 => these != the
387 => to != now
388 => be != he
389 => as != had
390 => as != and
391 => so != to
392 => and != a
393 => it != to
394 => in != and
395 => in != of
396 => if != to
397 => the != be
398 => or != for
399 => one != he
400 => to != a
401 => you != of
402 => to != you
403 => a != it
404 => s != is
405 => ios != his
406 => a != so
407 => an != so
408 => said != and
409 => in != of
410 => at != and
411 => said != and
412 => in != a
413 => an != man
414 => to != the
415 => not != you
```

```
416 => a != to
417 => on != to
418 => the != be
419 => one != the
420 => are != have
421 => than != the
422 => is != of
423 => in != and
424 => a != be
425 => in != of
426 => air != and
427 => in != and
428 => 3 != in
429 => in != and
430 => for != of
431 => ag != and
432 => in != of
433 => not != it
434 => as != and
435 => the != to
436 => a != way
437 => on != and
438 => due != the
439 => the != be
440 => on != to
441 => the != be
442 => no != a
443 => than != the
444 => the != than
445 => said != and
446 => its != it
447 => is != as
448 => to != the
449 => at != and
450 => 3 != to
451 => in != to
452 => the != be
453 => to != the
454 => its != is
455 => on != of
456 => this != his
457 => in != s
458 => can != and
459 => so != tv
460 => one != and
461 => of != k5
462 => u != he
463 => a != be
```

```
464 => the != other
465 => u != he
466 => law != a
467 => t != he
468 => in != by
469 => if != he
470 => was != is
471 => do != be
472 => u != of
473 => u != he
474 => had != and
475 => to != a
476 => hand != man
477 => the != to
478 => for != to
479 => the != be
480 => by != of
481 => to != in
482 => case != same
483 => are != the
484 => they != the
485 => have != he
486 => no != to
487 => s != he
488 => as != of
489 => in != as
490 => is != in
491 => in != to
492 => to != in
493 => act != a
494 => the != to
495 => said != had
496 => an != to
497 => in != of
498 => it != of
499 => for != to
500 => to != in
501 => act != a
502 => to != of
503 => take != the
504 => case != same
505 => for != not
506 => s != of
507 => is != he
508 => in != is
509 => on != to
510 => the != be
511 => the != he
```

```
512 => the != her
513 => to != of
514 => in != and
515 => are != at
516 => to != not
517 => is != in
518 => to != the
519 => the != to
520 => the != he
521 => the != be
522 => a != of
523 => to != the
524 => for != of
525 => a != to
526 => the != he
527 => u != he
528 => law != a
529 => no != to
530 => in != he
531 => their != the
532 => its != in
533 => or != of
534 => a != it
535 => more != for
536 => to != of
537 => what != that
538 => is != to
539 => it != to
540 => it != is
541 => is != it
542 => that != the
543 => buy != be
544 => as != of
545 => are != and
546 => buy != be
547 => an != a
548 => a != and
549 => t != be
550 => be != the
551 => of != it
552 => in != of
553 => end != and
554 => up != a
555 => buy != be
556 => it != i
557 => win != man
558 => in != and
559 => the != he
```

```
560 => if != i
561 => stop != ship
562 => it != of
563 => to != in
564 => in != a
565 => an != or
566 => this != the
567 => time != same
568 => s != of
569 => more != for
570 => t != be
571 => his != him
572 => this != the
573 => a != to
574 => to != a
575 => is != of
576 => as != if
577 => his != he
578 => in != and
579 => t != i
580 => rape != have
581 => has != had
582 => in != and
583 => way != man
584 => not != you
585 => more != are
586 => are != and
587 => as != and
588 => and != as
589 => is != he
590 => i != is
591 => it != i
592 => t != i
593 => as != and
594 => us != a
595 => it != i
596 => the != he
597 => the != he
598 => can != and
599 => t != a
600 => i != so
601 => it != i
602 => if != he
603 => re != were
604 => go != not
605 => was != had
606 => of != an
607 => as != of
```

608 => a != to
609 => he != is
610 => re != were
611 => he != is
612 => and != a
613 => re != were
614 => the != to
615 => i != a
616 => don != man
617 => t != s
618 => it != i
619 => a != to
620 => man != and
621 => he != i
622 => not != to
623 => all != and
624 => of != a
625 => as != and
626 => is != of
627 => there != the
628 => in != of
629 => s != is
630 => a != to
631 => there != the
632 => no != a
633 => do != of
634 => in != a
635 => in != of
636 => the != her
637 => she != he
638 => get != be
639 => the != to
640 => it != of
641 => a != of
642 => in != as
643 => or != of
644 => want != was
645 => to != a
646 => do != be
647 => a != to
648 => or != of
649 => a != if
650 => a != way
651 => them != the
652 => it != to
653 => man != and
654 => to != so
655 => in != and

```
656 => his != if
657 => don != in
658 => by != be
659 => and != a
660 => t != i
 ='( Ahora si )'=
```

[ ]: