

**CS491 – Natural Language Processing**

**ACADEMIC YEAR – 2021-2022**

**Semester - VII**

LAB – VI

BY:

ARJUN BAKSHI

BT18GEC134

CODE:

from spellchecker import SpellChecker

from tabulate import tabulate

#Utility functions below

def ngram(word):

splitword = [char for char in word]

res = [i + j for i, j in zip(splitword, splitword[1:])]

return res

def levenshteinDistance(s1, s2):

if len(s1) > len(s2):

s1, s2 = s2, s1

distances = range(len(s1) + 1)

for i2, c2 in enumerate(s2):

distances\_ = [i2+1]

for i1, c1 in enumerate(s1):

if c1 == c2:

distances\_.append(distances[i1])

else:

distances\_.append(1 + min((distances[i1], distances[i1 + 1], distances\_[-1])))

distances = distances\_

return distances[-1]

word = "machne"

wordbigram = ngram(word)

print("Bigrams: ", wordbigram)

spell = SpellChecker()

#print(spell.WordFrequency())

bestword = spell.correction(word)

print('Actual: ',spell.correction(word))

print('==================')

candidates = spell.candidates(word)

print("Candidate wrods below:")

for i in candidates:

print(i)

print('Candidate, Edit Distance')

for i in candidates:

print(i,' ',levenshteinDistance(word,i))

splitword = []

count = 0

collection = {}

for ele in candidates:

splitword = ngram(ele)

for i in splitword:

if i in wordbigram:

count = count+1

collection[ele] = count

count = 0

print('Candidate, Number of bigrams matching')

for i in collection:

print(i,' ',collection[i])

#Second function for finding minimum edit distance.

#First function was giving same edit distances for some words and hence we are unable to find out the actual word

#Hence Explored alternate function for finding edit distance so that only one word has the minimum edit distance

def editDistance(str1, str2, m, n):

if m == 0:

return n

if n == 0:

return m

if str1[m-1] == str2[n-1]:

return editDistance(str1, str2, m-1, n-1)

insertion = []

deletion = []

insertion.append(editDistance(str1, str2, m, n-1))

deletion.append(editDistance(str1, str2, m-1, n))

return 1 + min(editDistance(str1, str2, m, n-1), # Insert

editDistance(str1, str2, m-1, n), # Remove

editDistance(str1, str2, m-1, n-1) # Replace

)

splitword = []

count = 0

collection = {}

for ele in candidates:

splitword = ngram(ele)

for i in splitword:

if i in wordbigram:

count = count+1

collection[ele] = count

count = 0

print('Mispelled word is: ',word)

for i in collection:

str1 = word

str2 = i

tablelist = []

for num, letter in enumerate(i):

for letter2 in word:

if (num+1 < len(i) and num-1 >= 0):

if letter == letter2:

tablelist.append("match")

elif letter2 == i[num+1] and letter == letter2:

tablelist.append("match")

elif letter2 == i[num+1]:

new = "insert--"+letter

tablelist.append(new)

print(f"{i}: {tablelist}")

OUTPUT:

Bigrams: ['ma', 'ac', 'ch', 'hn', 'ne']

Actual: machine

==================

Candidate wrods below

machine

mache

machen

Candidate, Edit Distance

machine 1

mache 1

machen 2

Candidate, Number of bigrams matching

machine 4

mache 3

machen 3

Mispelled word is: machne

machine: ['match', 'insert--a', 'match', 'insert--c', 'match', 'insert--i', 'match', 'insert--n']

mache: ['match', 'insert--a', 'match', 'insert--c', 'match', 'insert--h']

machen: ['match', 'insert--a', 'match', 'insert--c', 'match', 'insert--h', 'insert--e', 'match']

CONCLUSION

1. It is observred that there can be a number of ways to calculate edit distance. Since some methods do give th same edit distance for some candidate words we cannot be sure as to which is the acutall and correct word. In the code output above the same is demonstrated since machne has two candidate words with same edit distance and hence makes us unable to give proper prediction.
2. Hence 2 methods can be used instead of one. Laso, the corpus used matters a lot. If if doesn’t cotain some essential words the spelling will not be corrected irrespective of how good the the edit distance function is.