----------SPELL_CHECKER_EN----------

SPELL CHECKER EN is a setting up a simple spell checking algorithm. It uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Dictionary was generated using the WordFrequency project on GitHub.

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

Mathematically, the Levenshtein distance between two strings a, b (of length |a| and |b| respectively) is given by leva,b(|a|,|b|) where:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Computing the Levenshtein distance is based on the observation that if we reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second, then we can compute the values in the matrix in a dynamic programming fashion, and thus find the distance between the two full strings as the last value computed.

This is a straightforward pseudocode implementation for a function LevenshteinDistance that takes two strings, s of length m, and t of length n, and returns the Levenshtein distance between them:

```
function LevenshteinDistance(char s[1..m], char t[1..n]):
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t
  declare int d[0..m, 0..n]

  set each element in d to zero

  // source prefixes can be transformed into empty string by
  // dropping all characters
  for i from 1 to m:
      d[i, 0] := i

  // target prefixes can be reached from empty source prefix
  // by inserting every character
  for j from 1 to n:
      d[0, j] := j

  for j from 1 to n:
      for i from 1 to m:
          if s[i] = t[j]:
            substitutionCost := 0
          else:
            substitutionCost := 1

          d[i, j] := minimum(d[i-1, j] + 1,                  // deletion
                             d[i, j-1] + 1,                  // insertion
                             d[i-1, j-1] + substitutionCost)  // substitution

  return d[m, n]
```

Installation

The easiest method to install is using pip:

pip install pyspellchecker

Additional Methods:

On-line documentation is available; below contains the cliff-notes version of some of the available functions:

correction(word): Returns the most probable result for the misspelled word

candidates(word): Returns a set of possible candidates for the misspelled word

known([words]): Returns those words that are in the word frequency list

unknown([words]): Returns those words that are not in the frequency list

word_probability(word): The frequency of the given word out of all words in the frequency list

It has three parts :

1. *A script.py file that takes input as a paragraph and gives a sequence of lines in their most appropriate words.It uses the module spellchecker to find out the correct words using function correction() to do so*

35 lines (28 sloc) | 1.16 KB

Raw Blame

```
from spellchecker import SpellChecker
import re

# creating spellchecker object
spell = SpellChecker()

# creating file object
f = open("input.txt", 'r')

# importing contents of input.txt file into data
data = f.read()

# dividing the contents of input.txt file into list of sentences with the delimiters being "." and "\n"
sentences = re.split('; |, |\.|\n', data)

# removing empty list elements formed from the above operation
while "" in sentences:
    sentences.remove("")
# print(sentences)

# for loop for looping over the list of sentences
for sentence in sentences:
    # creating a temp list for storing the corrected words
    correct_word = []
    # using strip() to clear any white spaces from the list element and them splitting them into individual words
    words = sentence.strip().split(" ")
    # print(words)
    # applying the correction function from the pyspellchecker library and then adding them to the temp list
    for word in words:
        correct_word.append(spell.correction(word))
    # finally joining the individual words and displaying the correctly spelled words to form a sentence
    print(" ".join(correct_word) + ".")

# closing the file
f.close()
```

.

2. *A GUI interface that takes multiple words ,sorts out multiple misspelled words and displays their most appropriate words . It uses the module tkinter to implement the GUI and also implements the spellchecker module to find out the corrected spellings.*

55 lines (36 sloc) | 1.6 KB

Raw Blame

```
from tkinter import *

from spellchecker import SpellChecker

spell = SpellChecker(distance=1)


def send():
    msg = EntryBox.get("1.0", 'end-1c').strip()

    EntryBox.delete("0.0", END)

    if msg != '':
        ChatLog.config(state=NORMAL)

        ChatLog.insert(END, "You: " + msg + '\n\n')

        ChatLog.config(foreground="#442265", font=("Verdana", 12))

        msg = msg.split(" ")
        misspelled = spell.unknown(msg)

        for word in misspelled:
            t = spell.candidates(word)
            n = spell.correction(word)
            ChatLog.insert(END, "Bot: Candidate words: " + str(t) + '\n\n')
            ChatLog.insert(END, "Bot: Probable word: " + str(n) + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)


base = Tk()
base.title("SPELL_CHECKER")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",  )
```

3 A CLI that takes a word and if it is misspelled then return the various appropriate words. It also makes use of spell checker module.

6 lines (4 sloc) | 120 Bytes

Raw Blame

```
from spellchecker import SpellChecker
import sys

spell = SpellChecker()

print(spell.candidates(str(sys.argv[1])))
```