Lesson 4

Lesson-End Project Solution



**Create Your Own Spell Checker**

**Objective:** Creating a spell checker, correct the incorrect word in the given sentence.

**Problem Statement:** While typing or sending any message to person, we generally make spelling mistakes. Write a script which will correct the misspelled words in a sentence. The input will be a raw string and the output will be a string with the case normalized and the incorrect word corrected.

**Domain:** General

**Analysis to be done:** Words availability in corpus

**Content:**Dataset: None

We will be using NLTK’s inbuilt corpora (words, stop words etc.) and no specific dataset.

**Steps to perform:**

While there are several approaches to correct spelling , you will use the Levenshtein or Edit distance approach .

The approach will be straightforward for correcting a word:

* If the word is present in a list of valid words, the word is correct.
* If the word is absent from the valid word list, we will find the correct word, i.e., the word from the valid word list which has the lowest edit distance from the target word.

Once you define a function, you will iterate over the terms in the given sentence, correct the words identified as incorrect, and return a joined string with all the terms. To help speed up execution, you won’t be applying the spell check on the stop words and punctuation.

**Tasks:**

1. Get a list of valid words in the English language using NLTK’s list of words (Hint: use nltk.download(‘words’) to get the raw list.

import nltk

nltk.download("words")

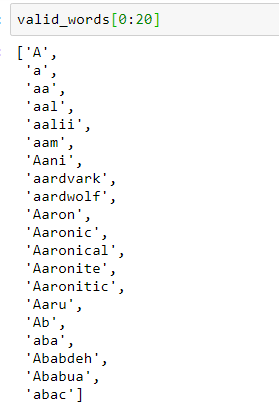
valid\_words = nltk.corpus.words.words()



1. Look at the first 20 words in the list. Is the case normalized?

valid\_words[0:20]

The words are not normalized in casing. So we’ll need to make sure all the words are of the same case, i.e., lower case.

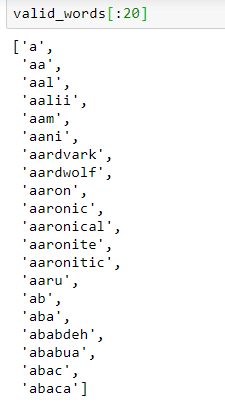


1. Normalize the casing for all the terms.

You can use the lower method for Python strings for this task.

valid\_words = [term.lower() for term in valid\_words]

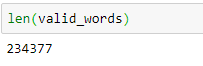
valid\_words[:20]



You can see that the casing has been successfully normalized.

1. Some duplicates would have been induced, create unique list after normalizing.

valid\_words = list(sorted(set(valid\_words)))



The length of the list has reduced by about 2300 words.

1. Create a list of stop words which should include:
   1. Stop words from NLTK.

from nltk.corpus import stopwords

stop\_nltk = stopwords.words("english")

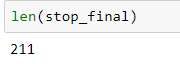
* 1. All punctuations (Hint: use ‘punctuation’ from string module).

from string import punctuation

stop\_punct = list(punctuation)

* 1. Final list should be a combination of these two**.**

stop\_final = stop\_nltk + stop\_punct



1. Define a function to get correct a single term
   1. For a given term, find its edit distance with each term in the valid word list. To speed up execution, you can use the first 20,000 entries in the valid word list.
   2. Store the result in a dictionary, the key as the term, and edit distance as value.

res\_dict = {valid\_term:nltk.edit\_distance(inp\_word, valid\_term) for valid\_term in valid\_words[:50000]}

* 1. Sort the dictionary in ascending order of the values.

res\_dict\_sorted = sorted(res\_dict.items(), key=lambda kv: kv[1], reverse=False)

* 1. Return the first entry in the sorted result (value with minimum edit distance).

return res\_dict\_sorted[0][0]

Final function:

def get\_correct\_term(inp\_word):

res\_dict = {valid\_term:nltk.edit\_distance(inp\_word, valid\_term) for valid\_term in valid\_words[:50000]}

res\_dict\_sorted = sorted(res\_dict.items(), key=lambda kv: kv[1], reverse=False)

return res\_dict\_sorted[0][0]

* 1. Using the function, get the correct word for committee.

get\_correct\_term("comittee")



You see that corrector does indeed find the right spelling for the term!

1. Make a set from the list of valid words, for faster lookup to see if word is in valid list or not.

valid\_words\_set = set(valid\_words)

1. Define a function for spelling correction in any given input sentence:
   1. To tokenize them after making all the terms in lowercase

inp\_tokens = word\_tokenize(inp\_sent.lower())

* 1. For each term in the tokenized sentence:
     1. Check if the term is in the list of valid words (valid\_words\_set).
     2. If yes, return the word as is.
     3. If no, get the correct word using get\_correct\_term function.

corrected\_tokens = [term if ((term in valid\_words\_set) or (term in stop\_final)) else get\_correct\_term(term) for term in inp\_tokens]

* 1. To return the joined string as output.

return " ".join(corrected\_tokens)

Final function:

def correct\_set(inp\_sent):

inp\_tokens = word\_tokenize(inp\_sent.lower())

corrected\_tokens = [term if ((term in valid\_words\_set) or (term in stop\_final)) else get\_correct\_term(term) for term in inp\_tokens]

return " ".join(corrected\_tokens)

1. Test the function for the input sentence “The new abacos is great”.

correct\_set('The new abacos is great')



You can see that with some simple rules and Edit or Levenshtein distance, you could build a basic spell checker. Of course, there are a lot of improvements which can be made for both efficiency and accuracy reasons.