

Implementation of the Spellchecker by Anubhab Dey

(Original idea by Peter Norvig)

What is the logic behind the spell checker?

The Spell checker uses a brute force approach to find possible correct words in upto 2 edit distances in a data set provided as a text file.

the Spell checker will consider any letter in an input incorrectly spelt word and:

- remove it
- swap it with a nearby letter
- add a letter before or after it
- and replace it with any other letter of the English alphabet.

While such a brute force approach is usually omitted in designing algorithms, it is okay to use it in case of spelling errors as most errors occur within 2 edit distances which is quite reasonable.

So we just need to pass the words from the first call into the working function to find all probable words in the data set provided.

How does the code work?

*the spellcheck uses Python 3

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

```
WORDS = Counter(words(open('big.txt', encoding="utf-8").read()))
```

Just reading input from the text file which we are using as a database. The database used is a sample essay. The words function uses findall and the regex used along with the findall function will separate all the different words. The counter function counts all the different instances of a word and saves it in a dictionary with key being the word and value being the instances of it occurring in the text sample.

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

```
def correction(word):  
    return max(candidates(word), key=P)
```

```
def candidates(word):  
    return (known([word]) or known(edits1(word)) or  
known(edits2(word)) or [word])
```

```
def known(words):  
    return set(w for w in words if w in WORDS)
```

P returns the probability of the word occurring in the text file. *Correction* returns a singular word which is the most probable correct word. *Candidates* return the list of all words present in the *WORDS* dictionary as *known*.

```

def edits1(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):
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))

```

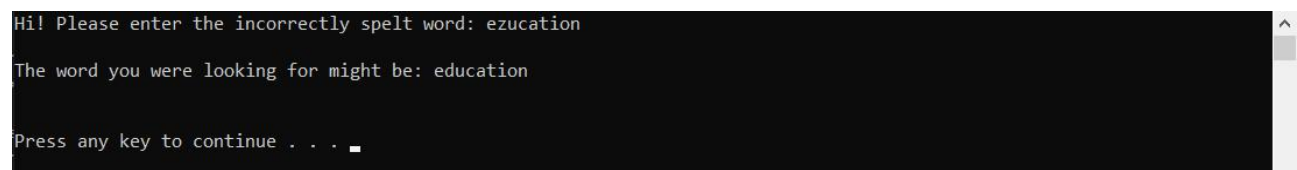
edits1 finds all the words by brute force. Deletes, swaps, replaces, inserts letters at all positions of the incorrect word. *edits2* uses the words found in *edits1* and does the same as passes it as input to *edits1* again.

Analytics in the project

Analytics is really essential for many NLP projects. As in the NLP project, analysing the sample essay is important to find

all the different words and how many times they occur. This is so that we can find all the different probable correct words and also find the probability of the correct word being the one the user is asking for. Analysing the data is therefore essential to find an optimal solution to many NLP related problems.

Exemplar Execution snippet:

A terminal window with a dark background and light gray text. It shows a sequence of three lines: a prompt asking for an incorrectly spelt word, a suggestion of the correct word, and a prompt to press a key to continue.

```
Hi! Please enter the incorrectly spelt word: ezucation
The word you were looking for might be: education
Press any key to continue . . . █
```