In [1]:

```
# regular expression libreary
import re
# Alternative of Decontraction libreary(Faced Java issue)
import contractions
from textsearch import TextSearch

import numpy as np
# for calculate weighted levenshtein
from weighted_levenshtein import lev, osa, dam_lev
# nltk libreary for biagrams
from nltk import word_tokenize, ngrams

# this function sepreates sentences with .,!,?,\n regular expression.
def listSplit(name):
    file = open(name, "r")
    doclist = [ line.lower() for line in file ]
    docstr = ''. join(doclist)
    sentences = re.split(r'[.!?\n]', docstr)
    sentences = [x for x in sentences if x != '']
    return sentences

# Add contractions in sentances if necessary . Eg: I'd -> I would
def expand_contractions(sens):
    deContraction = []
    for i,j in enumerate(sens):
        deContraction.append(contractions.fix(j))
    return deContraction

# Seperate tokens from each sentences.
def token_sept(sens):
    tokens= []
    for i,j in enumerate(sens):
        tokens.append(j.split(' '))
    return tokens
```

In [2]:

```
# pick all words from corpus whose size is (len(recognised_word) -1) or len(recognis
# For example my rec_word is : hello len is 5. Then all letters from corpus of size
def picked_words(expand_contractions,lenWord):
    counts_word = []
    for d in expand_contractions:
        for j in d:
            if len(j) >= lenWord-1 and len(j) <= lenWord+1:
                counts_word.append(j)
    return counts_word
```

In [3]:

```
# Typographic errors detection refered "https://pdfs.semanticscholar.org/c64f/1bd3a
def edits1(word):
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    split = []
    deletes = []
    transposes = []
    replaces = []
    inserts = []

    for i in range(len(word)+1):
        split.append([word[:i], word[i:]])

    for i,j in split:
        if j:
            deletes.append(i + j[1:])

    for i,j in split:
        if len(j)>1:
            transposes.append(i + j[1] + j[0] + j[2:])

    for i,j in split:
        if j:
            for c in letters:
                replaces.append(i + c + j[1:])


    for i,j in split:
        for c in letters:
            inserts.append(i + c + j)


    return list(deletes+inserts+transposes+replaces)
```

In [4]:

```
# check all possiblities of Typographic errors in to the word picked.
def result(final_select,picked_words):
    data = []
    for i in final_select:
        if i in picked_words:
            data.append(i)
    return data
```

In [5]:

```
nshtein with characted differences.
rd):
nes((128, 128), dtype=np.float64)

y be : {} the change in char is of  : {} digits".format(i,int(dam_lev(recWord, i, sul
```

In [ ]:

```

```

In [6]:

```
# biagram function. Eg. 'Hello' -> ['he','el','ll','lo']
def ngram(s1):
    return list(ngrams(s1, 2))
```

In [7]:

```
# remove doubles into the biagram for calcuation purpus. refered in the same researc
def remove_double(s1):
    count = 0
    for (filename,filepath) in s1:
        count = count + 1
        for (filename1,filepath1) in s1[count:]:
            if filename == filename1:
                s1.remove((filename1,filepath1))
    return s1
```

In [8]:

```
# Accuracy count for 2 biagram.
def accuracy_count(s1,s2):
    data = []
    for i,j in s1:
        for k,l in s2:
            if i==k and j==l:
                data.append((i,j))
    return (2*len(data)) / (len(s1)+len(s2))
```

In [ ]:

```

```

In [9]:

```
# data fetch
sentences = listSplit('europarl-v7.de-en.en')

# sentences creations.
sentences = sentences[0:len(sentences)-1]

# identifie contractions.
expand_contrac = expand_contractions(sentences)

# convert sentennces to tokens.
tokens = token_sept(expand_contrac)
```

In [ ]:

```

```

In [13]:

```
recWord = 'little'
```

In [14]:

```
# pick all words from corpus according sto size.
picked_words=picked_words(tokens,len(recWord))

# checking all possiblities of Typographic errors
final_selection=edits1(recWord)

# results with uniquness between recognised word and corpus word
answer = result(final_selection,picked_words)

# print the result with Weighted Levenshtein
data_print(answer,recWord)
# if the multiple answer is found then gives the output of all multiple word occura
for i in answer:
    s1_N = ngram(i)
    s2_N = ngram(recWord)

    s1_rm = remove_double(s1_N)
    s2_rm = remove_double(s2_N)

    print('Biagram --> word from corpus = {} and recognises = {} and accuracy is =
```

In [18]:

```
# Test case 1:
    #  Biagram --> word from corpus = unetek and recognises = unitek and accuracy is

# Test Case 2 :
    # Biagram --> word from corpus = little and recognises = little and accuracy is
    # Biagram --> word from corpus = little and recognises = tittle and accuracy is
```

In [ ]:

```

```