CMPE 493

Introduction to Information Retrieval

Spelling Error Correction

Şadi Uysal

2015400162

(i) Describe how you implemented the spelling error corrector.

I created my dictionary using the provided *corpus.txt* file. This file was obtained from Peter Norvig's web site (http://norvig.com/big.txt). It contains a concatenation of several public domain books from *Project Gutenberg* as well as lists of most frequent words from *Wiktionary* and the *British National Corpus*. I assumed that the words in the corpus.txt file are spelled correctly. In order to create my dictionary, I used regex and parsed the corpus file as **WORD, NEWLINE, ABBREVIATION** token types. Then for each token, performed case-folding and added corresponding token to my token dictionary with words as key and their frequencies as value. I also created bigram index dictionary to eliminate some of the word candidates.To achieve that, I used bigram's values as key and their corresponding words as values. I also fulfilled possible error's "count" values as channel model probability denominater in lists named singleCharCounts,doubleCharCounts. Then, I initialized confusion matricies with all zeros. I completed those matrices with "spell-errors.txt" file via computing needed Damerau-Levenshtein edits.

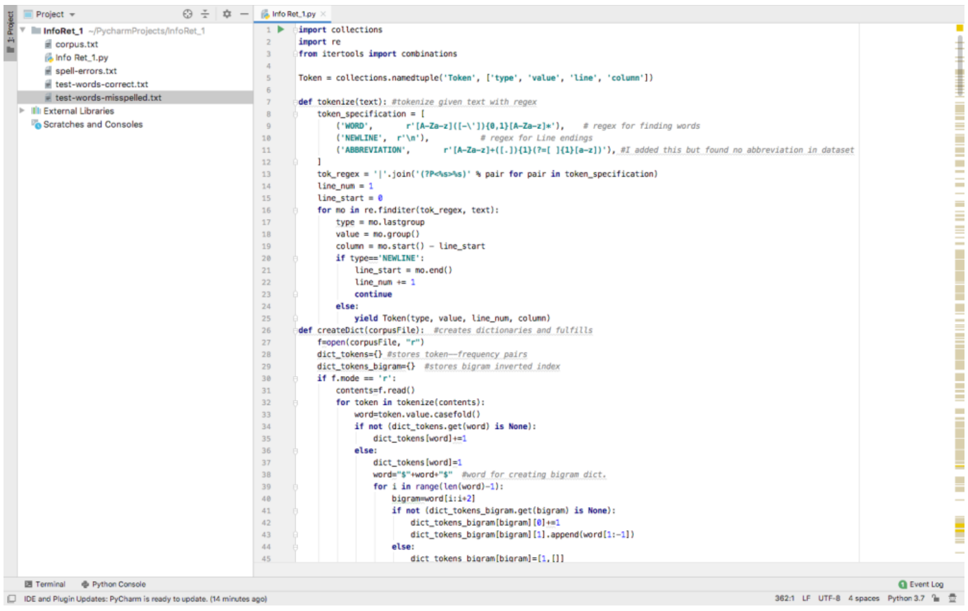
In the processing query step,for any given misspelled query, I do the following:

- Checked my token dictionary, if query already exists then query would not be misspelled then return query itself.
- If does not exists, we would not want to compute edit distances for all token dictionary.So I generated bigram set for given query and intersect needed bigram sets to get candidate words according to bigram Index. (bigramIndexCandidates())
- I get the bigramIndex candidates then dropped candidates with edit distances greater than 1.
- $w\hat{} = \text{argmax } P(x|w)\ P(w)$
- The only thing left is calculating above probabilities for each of the candidate word and get word with maximum probability.
- I compute the prior probability(P(w)) via token dictionary by summing up word frequencies.
- Then for each candidate, I analyzed edit distance matricies for finding needed edits and get corresponding counts from confusion matricies.
- I found candidate with maximum probability and wrote it to file named "correctedWords.txt"

(ii) Provide the four confusion matrices that you computed.

I added four matricies as text files in the matrices folder.

(iii) Provide screenshots of running your system.



```
Project
InfoRet_1 ~/PycharmProjects/InfoRet_1
    corpus.txt
    Info Ret_1.py
    spell-errors.txt
    test-words-correct.txt
    test-words-misspelled.txt
External Libraries
Scratches and Consoles
```

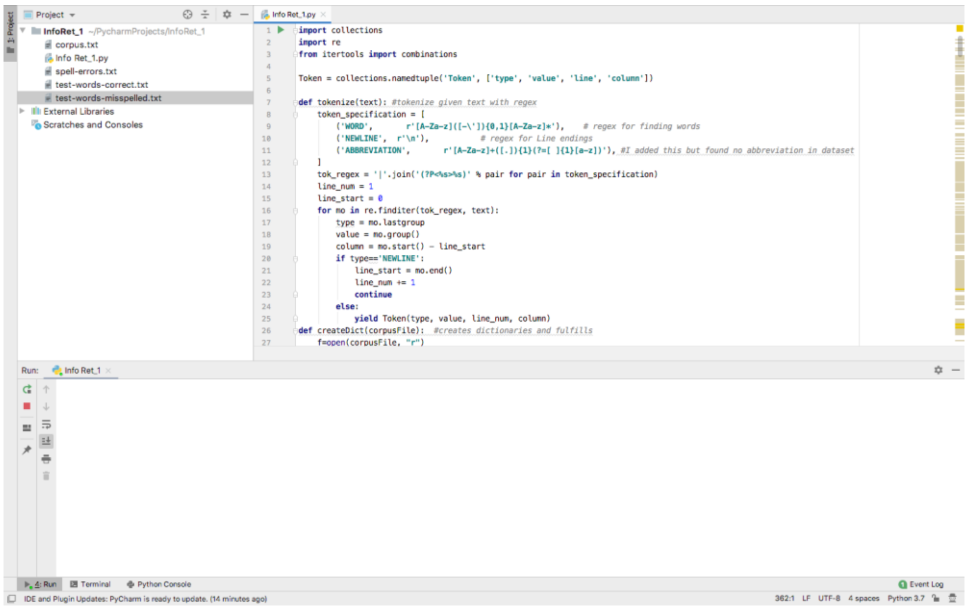
Info Ret_1.py

```python
import collections
import re
from itertools import combinations

Token = collections.namedtuple('Token', ['type', 'value', 'line', 'column'])

def tokenize(text): #tokenize given text with regex
    token_specification = [
        ('WORD',       r'[A-Za-z]([-\'']){0,1}[A-Za-z]*'),    # regex for finding words
        ('NEWLINE',  r'\n'),           # regex for Line endings
        ('ABBREVIATION',       r'[A-Za-z]+([.]){1}(?=[ ]{1}[a-z])'), #I added this but found no abbreviation in dataset
    ]
    tok_regex = '|'.join('(?P<%s>%s)' % pair for pair in token_specification)
    line_num = 1
    line_start = 0
    for mo in re.finditer(tok_regex, text):
        type = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if type=='NEWLINE':
            line_start = mo.end()
            line_num += 1
            continue
        else:
            yield Token(type, value, line_num, column)
def createDict(corpusFile):  #creates dictionaries and fulfills
    f=open(corpusFile, "r")
    dict_tokens={} #stores token--frequency pairs
    dict_tokens_bigram={}  #stores bigram inverted index
    if f.mode == 'r':
        contents=f.read()
        for token in tokenize(contents):
            word=token.value.casefold()
            if not (dict_tokens.get(word) is None):
                dict_tokens[word]+=1
            else:
                dict_tokens[word]=1
                word="$"+word+"$"  #word for creating bigram dict.
                for i in range(len(word)-1):
                    bigram=word[i:i+2]
                    if not (dict_tokens_bigram.get(bigram) is None):
                        dict_tokens_bigram[bigram][0]+=1
                        dict_tokens_bigram[bigram][1].append(word[1:-1])
                    else:
                        dict_tokens_bigram[bigram]=[1,[]]
```

```
Terminal    Python Console                                                    Event Log
IDE and Plugin Updates: PyCharm is ready to update. (14 minutes ago)      362:1  LF  UTF-8  4 spaces  Python 3.7
```



```
Project
InfoRet_1 ~/PycharmProjects/InfoRet_1
    corpus.txt
    Info Ret_1.py
    spell-errors.txt
    test-words-correct.txt
    test-words-misspelled.txt
External Libraries
Scratches and Consoles
```

Info Ret_1.py

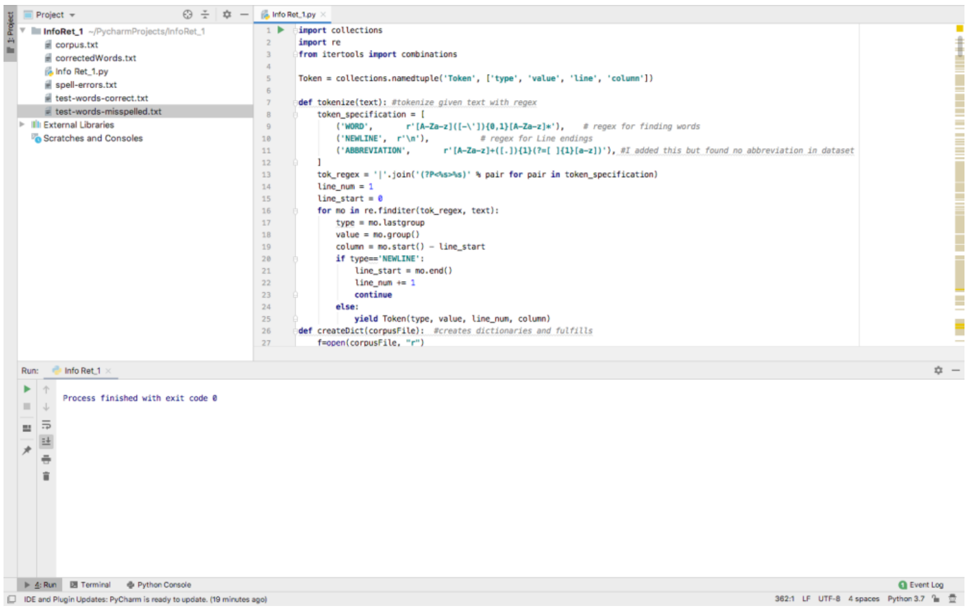
```python
import collections
import re
from itertools import combinations

Token = collections.namedtuple('Token', ['type', 'value', 'line', 'column'])

def tokenize(text): #tokenize given text with regex
    token_specification = [
        ('WORD',       r'[A-Za-z]([-\'']){0,1}[A-Za-z]*'),    # regex for finding words
        ('NEWLINE',  r'\n'),           # regex for Line endings
        ('ABBREVIATION',       r'[A-Za-z]+([.]){1}(?=[ ]{1}[a-z])'), #I added this but found no abbreviation in dataset
    ]
    tok_regex = '|'.join('(?P<%s>%s)' % pair for pair in token_specification)
    line_num = 1
    line_start = 0
    for mo in re.finditer(tok_regex, text):
        type = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if type=='NEWLINE':
            line_start = mo.end()
            line_num += 1
            continue
        else:
            yield Token(type, value, line_num, column)
def createDict(corpusFile):  #creates dictionaries and fulfills
    f=open(corpusFile, "r")
```

```
Run:  Info Ret_1
```

```
4: Run    Terminal    Python Console                                          Event Log
IDE and Plugin Updates: PyCharm is ready to update. (14 minutes ago)      362:1  LF  UTF-8  4 spaces  Python 3.7
```



```python
import collections
import re
from itertools import combinations

Token = collections.namedtuple('Token', ['type', 'value', 'line', 'column'])

def tokenize(text): #tokenize given text with regex
    token_specification = [
        ('WORD',       r'[A-Za-z]([-\'']){0,1}[A-Za-z]*'),    # regex for finding words
        ('NEWLINE',  r'\n'),           # regex for Line endings
        ('ABBREVIATION',       r'[A-Za-z]+([.]){1}(?=[ ]{1}[a-z])'), #I added this but found no abbreviation in dataset
    ]
    tok_regex = '|'.join('(?P<%s>%s)' % pair for pair in token_specification)
    line_num = 1
    line_start = 0
    for mo in re.finditer(tok_regex, text):
        type = mo.lastgroup
        value = mo.group()
        column = mo.start() - line_start
        if type=='NEWLINE':
            line_start = mo.end()
            line_num += 1
            continue
        else:
            yield Token(type, value, line_num, column)
def createDict(corpusFile):  #creates dictionaries and fulfills
    f=open(corpusFile, "r")
```

```
Process finished with exit code 0
```

(iv) Report the accuracy scores you obtained for the provided test set both by using smoothing and without smoothing.

I checked my errors and realized that some of the misspelled words have more than 1 edit distance to their correct form.(18 words) Also our corpus.txt file do not have some of correct words so that I could include them to my word dictionary. (60 words)

With above ones excluded in the test set, I had accuracy: 92.8%

With above ones included in the test set, I had accuracy: 74%

I had same accuracy with smoothing also.So, I think we do not have unseen errors for this test set.

(v) Investigate the errors of your system and discuss how your system can be improved.

As I mentioned above, since our corpus was not enough to represent all possible corrected words I could not find some of the correct words since they are not in my token dictionary. Besides, spelling-errors data set was not enough to represent all possible errors so their probabilities. In example, there is no abbreviation in the corpus file, so their usage count would be zero but we need them to estimate probability because they exists in spell-errors file. We can improve system with greater data-sets. Furthermore, I would do better tokenization since small portion of tokens did not make sense to me. And lastly, we can give different weights to errors to improve accuracy.