IRTM,
Homework 2

Dennis Tschechlov
Felix Bühler
Tobias Boceck
Manuel Mergl

22. November 2017

Programming Task - SubTask 2

New Lines are marked blue!

```
#!/usr/bin/env python3
from collections import defaultdict
import numpy as np
import sys
import re
name = ""
inv_index = defaultdict(list)
correct_spelling = defaultdict()
postings = defaultdict(set)
suggestions = defaultdict(set)
stop_words = {'a', 'an', 'and', 'are', 'as', 'at', '
   be', 'by', 'for', 'from',
              'has', 'he', 'in', 'is', 'it', 'its',
                 'of', 'on', 'that', 'the',
              'to', 'was', 'were', 'will', 'with'}
count = 0
# if you look on the keyboard, these are the keys
   around every key (US-Layout)
key_q = {"w", "s", "a"}
key_w = {"q", "e", "a", "s", "d"}
key_e = {"w", "r", "s", "d", "f"}
```

```
key_r = {"e", "t", "d", "f", "g"}
key_t = {"r", "y", "f", "g", "h"}
key_y = {"t", "g", "h", "j", "u"}
key_u = {"y", "i", "h", "j", "k"}
key_i = {"u", "o", "j", "k", "l"}
key_o = {"i", "p", "k", "l"}
key_p = {"o", "l"}
key_a = {"q", "w", "s", "z", "x"}
key_s = {"q", "w", "e", "a", "d", "z", "x", "c"}
key_d = {"w", "e", "r", "s", "f", "x", "c", "v"}
key_f = {"e", "r", "t", "d", "g", "c", "v", "b"}
key_g = {"r", "t", "y", "f", "h", "v", "b", "n"}
key_h = {"t", "y", "u", "g", "j", "b", "n", "m"}
key_j = {"y", "u", "i", "h", "k", "n", "m"}
key_k = {"u", "i", "o", "j", "l", "m"}
key_l = {"i", "o", "p", "k"}
key_z = {"a", "s", "x"}
key_x = {"a", "s", "d", "z", "c"}
key_c = {"s", "d", "f", "x", "v"}
key_v = {"d", "f", "g", "c", "b"}
key_b = {"f", "g", "h", "v", "n"}
key_n = {"g", "h", "j", "b", "m"}
key_m = {"h", "j", "k", "n"}

def normalize(term):
    """
    normalize word and remove useless stuff
    """
    return term.lower().\
        replace(":", "").\
        replace(";", "").\
        replace(".", "").\
        replace(",", "").\
        replace("/", "").\
        replace("#", "").\
        replace("!", "").\
        replace("?", "").\
        replace("’", "").\
        replace("\"", "")
def index(filename):
    """
```

```
indexes a given file and saves terms to a
   posting and
non-positional inverted index
"""
global name
name = filename
try:
     # open file
     with open(filename, "r") as file:
          docID = 0
          # iterate over each line in file
          for line in file:
               # split them to list of terms
               tweet = line.split()
               for term in tweet:
                    # remove clutter
                    term = normalize(term)
                    # check if term is in stop words
                         to save some memory
                    if not str.isalpha(term) or (
                       term in stop_words):
                         continue
                    # print(term)
                    addSuggestions(term)
                    # add docID
                    postings[term].add(docID)
                    # update inv_index
                    # we use the term as pointer,
                       because python does not
                    # support pointers, and storing
                       int by indexes will have an
                    # massive overhead
                    inv_index[term] = (len(postings[
                       term]), term)
               # increase line number counter
               docID += 1
               # this is for displaying a progress
                  while indexing
               if docID % 10000 == 0:
                    sys.stdout.write("\r{0}␣%".
                       format(int(int(docID) /
                     10000)))
                 sys.stdout.flush()
             # if docID >= 100000:
             #     break
    except FileNotFoundError as e:
        raise SystemExit("Could␣not␣open␣file:␣" +
           str(e))
    return
def getLines(lines):
    """
    get lines of multiple lines
    """
    result = ""
    try:
        # open file
        with open(name, "r") as file:
            # iterate over the lines and print the
               lines, which match the terms
            for i, line in enumerate(file):
                if i in lines:
                    result += str(i) + "\t" + line
    except FileNotFoundError as e:
        raise SystemExit("Could␣not␣open␣file:␣" +
           str(e))
    return result
def query(term1, term2=""):
    """
    you can query your search terms. If only one
       term given it only searches
    for one, otherwise they both have to exist in
       the tweet
    """
    lines = []
    # remove clutter
    term1 = normalize(term1)
    term2 = normalize(term2)
    # if only one term given look for it
    if term1 in inv_index and not term2:
        (postings_len, postings_pointer) = inv_index
           [term1]
    # the sorted document_id list out of the
       postings_list
    lines = list(postings[postings_pointer])
# if two terms are given look for both
elif term1 in inv_index and term2 and term2 in
   inv_index:
    (postings_len, postings_pointer) = inv_index
       [term1]
    # the sorted document_id list out of the
       postings_list
    lines1 = sorted(list(postings[
       postings_pointer]))
    (postings_len, postings_pointer) = inv_index
       [term2]
    # the sorted document_id list out of the
       postings_list
    lines2 = sorted(list(postings[
       postings_pointer]))
    # init of iterators
    listiter1 = iter(lines1)
    listiter2 = iter(lines2)
    tmp1 = -1
    tmp2 = -1
    # and intersect the lists to see which lines
        match both terms
    # if the have the same lines, it will be
       added to 'lines'
    while True:
        try:
            # like discused in the lesson
            if tmp1 <= tmp2:
                tmp1 = next(listiter1)
            else:
                tmp2 = next(listiter2)
            if tmp1 == tmp2:
                lines.append(tmp1)
        except StopIteration:
            break
else:
    # if nothing is found it will look for
       alternative querys in the
```

```
        # suggestions dict
        print("nothing␣found")
        # if simple query has nothing found
        if not term2:
            for new in suggestions[term1]:
                print("Possible␣search␣query:␣" +
                   new)
                lines += query(new)
        # if complex query has nothing found for
           both querys
        elif term1 in suggestions and term2 in
           suggestions:
            for new1 in suggestions[term1]:
                for new2 in suggestions[term2]:
                    print("Possible␣search␣query:␣"
                       + new1 + ",␣" + new2)
                    lines += query(new1, new2)
        # if complex query has nothing found for one
            query
        elif term1 in suggestions and term2 not in
           suggestions:
            print("Possible␣search␣query:")
            for new in suggestions[term1]:
                print("Possible␣search␣query:␣" +
                   new + ",␣" + term2)
                lines += query(new, term2)
        # if complex query has nothing found for the
            other one query
        elif term1 not in suggestions and term2 in
           suggestions:
            print("Possible␣search␣query:")
            for new in suggestions[term2]:
                print("Possible␣search␣query:␣" +
                   term1 + ",␣" + new)
                lines += query(term1, new)
    # return lines
    return lines
def read_correct(filename):
    """
    fill correct_spelling
    """
    try:
        # open file
        with open(filename, "r") as file:
            # iterate over the lines and add them to
                the correct_spelling
            for term in file:
                term = normalize(term)
                if term in stop_words:
                    continue
                if term not in correct_spelling:
                    correct_spelling[term] = 0
    except FileNotFoundError as e:
        raise SystemExit("Could␣not␣open␣file:␣" +
           str(e))
def addSuggestions(term):
    """
    fill suggestions but only generate worde with
       levinstein distance of one
    otherwise the programm will use to much ram
    with this configuration it will already use 33gb
        of ram
    """
    # iterate over the word by its lenght
    for i in range(0, len(term)):
        # get char at position i
        alpha = term[i]
        # if char ist not between a to z it skips to
            the next char
        if not re.match('[a-z]', alpha):
            continue
        # other wise it loads the keys around that
           char
        keys = eval('key_' + alpha)
        # it changes the char to every other value
           and stores it in suggestions
        for k in keys:
            new = list(term)
            new[i] = k
            suggestions[''.join(new)].add(term)
        # then it removes the one value and stores
           it
```

```
            new = list(term)
            del new[i]
            suggestions[''.join(new)].add(term)
            # adds key to the char (slipping of a key)
            new = list(term)
            for k in keys:
                new.insert(i, k)
                suggestions[''.join(new)].add(term)
            # adds keys to the position (dubble pressing
               )
            new = list(term)
            new.insert(i, alpha)
            suggestions[''.join(new)].add(term)
def levenshtein(A, B, thresh, insertion, deletion,
    substitution):
    """
    implementation of damerau-levenshtein from:
    https://gist.github.com/kylebgorman/1081951
    """
    D = np.zeros((len(A) + 1, len(B) + 1), dtype=np.
        int)
    for i in range(len(A)):
        D[i + 1][0] = D[i][0] + deletion
    for j in range(len(B)):
        D[0][j + 1] = D[0][j] + insertion
    for i in range(len(A)):  # fill out middle of
        matrix
        for j in range(len(B)):
            if A[i] == B[j]:
                D[i + 1][j + 1] = D[i][j]  # aka, it
                    's free.
            else:
                D[i + 1][j + 1] = min(D[i + 1][j] +
                    insertion,
                                      D[i][j + 1] +
                                          deletion,
                                      D[i][j] +
                                          substitution
                                          )
            if D[i + 1][j + 1] >= thresh:
                return None
    return D.item((-1, -1))
if __name__ == '__main__':
    read_correct("english-words")
    # print("finished reading words")
    index("tweets")
    # print("\nfinished indexing")
    # print(len(inv_index))
    # for right exit code
    try:
        # asking for more querys
        while True:
            # ask for input
            search = input('What␣are␣you␣looking␣for
               ?:␣')
            try:
                # if it is a simple query this is
                   successfull
                term1, term2 = search.split("␣")
            except ValueError:
                # if the line before fails only this
                    one will get executed
                # (simple query)
                print("simple␣search␣query")
                print(getLines(query(search)))
                continue
            # complex query
            print("tuple␣search␣query")
            print(getLines(query(term1, term2)))
    except KeyboardInterrupt:
        pass
```

Sample Output:

```
What are you looking for?: gelderm sejioren
tuple search query
nothing found
Possible search query: geldern, senioren
73940   2016-04-20 00:05:17 +0200
   722546640238874625      @rpo_geldern    RP Online
    Geldern       Geldern - Senioren machen eine
   Reise zurück in die Kinderzeit https://t.co/57
   qoWYf6LO
```

```
What are you looking for?: gelix
simple search query
nothing found
Possible search query: helix
Possible search query: felix
Possible search query: relix
...
What are you looking for?: watterfall
simple search query
nothing found
Possible search query: waterfall
...
What are you looking for?: un sttgart
tuple search query
nothing found
Possible search query: uni, stuttgart
Possible search query: uno, stuttgart
Possible search query: uno, sttutgart
... and much more results
```