Abstracts (encoding and decoding of text files)

Sam Jongman, Huub Visser

November 7, 2019

Abstract

We have made a program that encodes a text file per line into numbers, whilst keeping special character intact.

1 Encoding

After the user has chosen a file, they have the option to process it and save the encoded file as a text file and save the index file. The index file is a CSV-file in which the translation (corresponding letters to the words), frequency of the words within the whole file and the amount of times a word appears in a line (abstract) is written. When it encodes a file it does it by reading the text file line by line. Each different word gets a different number assigned to it so if a word appears multiple times in a file it has the same number assigned to it. It also creates a textual histogram of the amount of times a word begins with a certain letter.

2 Decoding

Unfortunately, we did not have enough time to program a decoder and it also doesn't calculate the TF/IDF-scores of a line.

3 Source code listing

Below you will find the listing of our program. We have tried to use the suggestions in the assigneme

```
print('''
Student .......: Sam Jongman, Huub Visser
Number ........: s2550040, s2568861
Assignment ....: Abstracts
Last Edit Date : Nov, 7 2019
Description ...: This program encodes and decodes text files that the user ca
''')

import os
import math

def main_menu(selected_file):
                                                    # This asks what the user
    clear_console();
    options = "1"
    print('Welcome␣to␣Abstracts')
    print('')
    print('This␣program␣is␣used␣to␣encode␣or␣decode␣text␣files.')
    print('')
    if encode_or_decode(selected_file) == 'encode':
        print('Selected␣file:␣' + selected_file)
        print('Output␣file:␣' + get_output_file(selected_file))
        print('Index␣file:␣' + get_index_file(selected_file))
    elif encode_or_decode(selected_file) == 'decode':
        print('Selected␣file:␣' + selected_file)
        print('Output␣file:␣' + get_output_file(selected_file))
    else:
        print('No␣.txt␣or␣index.csv␣file␣selected.')
    print('')
    print('1:␣Select␣a␣file')
    if selected_file.lower().endswith('txt'): #This checks what kind of file
        print('2:␣Encode␣' + selected_file)
        options += ",␣2"
    if selected_file.lower().endswith('csv'):
        print('2:␣Decode␣' + selected_file)
        options += ",␣2"
    print('0:␣Exit␣program')
    print('')
    return int(input('Please␣type␣' + options + '␣or␣0:␣'))

def get_output_file(file):
    if encode_or_decode(file) == "encode":
        return file + '.encoded.txt'
    return file.replace('.index.csv', '.decoded.txt') #this creates a filenam

def get_index_file(file):
    if encode_or_decode(file) == "encode":   #this creates a filename for a s
        return file + '.index.csv'

def encode_or_decode(selected_file):
    if selected_file.lower().endswith('txt'): #this checks if a the selected
```

```
        return 'encode'
    if selected_file.lower().endswith('index.csv'):
        return 'decode'
    return 'wrong␣file'

def clear_console():
    print("\n" * 100) #prints some white lines

def file_menu():
    clear_console();
    print('')
    print('Select␣a␣.txt␣file␣to␣encode␣or␣a␣.index.csv␣file␣to␣decode')
    print('')
    print('The␣current␣directory␣is␣' + os.getcwd()) # checks current directo
    print('')
    menu = 1
    print('1:␣../')
    files = {0:'', 1: '../'}
    for entry in os.listdir(os.getcwd()):
        if not entry.startswith('.'):
            menu = menu + 1
            if os.path.isdir(entry):
                entry = entry + '/'
            print(str(menu) + ':␣' + entry)
            files[menu] = entry
    print('0:␣Exit␣program')
    print('')
    choice = int(input('Please␣type␣1␣to␣' + str(menu) + '␣or␣0:␣')) # with t
    return files[choice]

def select_file():
    file_name = file_menu()
    if file_name.endswith('/'): #if it is another directory it goes in there
        os.chdir(file_name)
        return select_file()
    if len(file_name) == 0:
        return ''
    return file_name

def count_word_in_abstracts(word, abstr_freqs): #how many words are in an abs
    count = 0
    for abstract_index in abstr_freqs:
        if word in abstr_freqs[abstract_index]:
            count += 1
    return count
```

```
def write_decoded_files(selected_file, words, codes, abstr_freqs,
                        total_freqs, abstracts, encodedAbstracts): #unfortunatly
    print("to_do")

def write_encoded_files(selected_file, words, codes, abstr_freqs, #this write
                        total_freqs, abstracts, encodedAbstracts):
    output_file = get_output_file(selected_file)
    file = open(output_file, "w+")
    for abstract_index in encodedAbstracts:
        file.write(encodedAbstracts[abstract_index] + '\n')
    file.close()

    csv_file = get_index_file(selected_file) #this writes the csv file
    file = open(csv_file, "w+")
    file.write("word,number,frequency,abstracts\n") #title of csv file

    for word in codes:
        if codes[word] == 0:
            continue
        file.write(word + ',')
        file.write(str(codes[word]) + ',')
        if word in total_freqs:
            file.write(str(total_freqs[word]) + ',')
            file.write(str(count_word_in_abstracts(word, abstr_freqs)) + '\n'
        else:
            file.write("0,")
            file.write("0\n")

    for word in codes:
        if codes[word] != 0:
            continue
        file.write(word + ',')
        file.write(str(codes[word]) + ',')
        if word in total_freqs:
            file.write(str(total_freqs[word]) + ',')
            file.write(str(count_word_in_abstracts(word, abstr_freqs)) + '\n'
        else:
            file.write("0,")
            file.write("0\n")

    file.close()

def print_encode_result(selected_file, words, codes, total_freqs, abstracts):
    orig_size = os.path.getsize(selected_file);
    output_size = os.path.getsize(get_output_file(selected_file));
    index_size = 0
    for word in codes:
        index_size += len(word)

    comp_rate = math.floor((output_size + index_size) / orig_size * 100) #com
    print()
    print("————————————————————————————————")
    print("Found␣" + str(len(words)) + "␣unique␣words")
#this prints the amount of unique words found in how many abstracts
    print("in␣" + str(len(abstracts)) + "␣abstracts.")
    print("The␣compression␣rate␣is␣" + str(comp_rate) + "%")
    print("————————————————————————————————")
    print()

    words_per_letter = {}
    unique_words_per_letter = {}                    # this creates dicts
    for letter in "abcdefghijklmnopqrstuvwxyz":
        words_per_letter[letter] = 0
        unique_words_per_letter[letter] = 0
        for word in total_freqs:
            if word[:1] == letter:
                if word.lower() == word:
                    words_per_letter[letter] += total_freqs[word]
                    unique_words_per_letter[letter] += 1

    all_players = {}
    for letter in "abcdefghijklmnopqrstuvwxyz":
        all_players[letter] = letter

    weener_list = {}
    while len(weener_list) < 26:
        weener = next(iter(all_players))
        weener_freq = words_per_letter[weener]
        for letter in all_players:
            if letter == weener:
                continue
            if words_per_letter[letter] > weener_freq:
                weener = letter
                weener_freq = words_per_letter[letter]
        del all_players[weener]
        weener_list[weener] = weener_freq
    for letter in weener_list:
        most_words = words_per_letter[max(words_per_letter, key=words_per_let
        bars_equal = int(round(words_per_letter[letter] / most_words * 40))
        bars_plus = int(round(unique_words_per_letter[letter] / most_words *
#this calculates how many + and = has to be printed in the textual histogram
```

```
        amount_bars_equal = ''
        amount_bars_plus = ''
        count_equal = 0
        count_plus = 0
        while bars_plus > count_plus:
            amount_bars_plus = amount_bars_plus + '+'
            count_plus = count_plus + 1
        while bars_equal - count_plus > count_equal :
            amount_bars_equal = amount_bars_equal + '='
            count_equal = count_equal + 1

        fmt = '{: <40}'.format(amount_bars_plus + amount_bars_equal)
        print(letter + ' | '  + fmt, end='')
        fmt = '{:>5}'.format(unique_words_per_letter[letter])
# this makes the histogram orginized and places spaces if there are no + or =
        print(' | ' + fmt  , end='')
        fmt = '{:>5}'.format(words_per_letter[letter])
        print(' | ' + fmt + ' | ' )


def main(selected_file):
    words, codes, abstr_freqs, total_freqs = {}, {}, {}, {}
    abstracts, encodedAbstracts = {}, {}

    while True:
        choice = main_menu(selected_file)

        if choice == 1:
            selected_file = select_file()
            if len(selected_file) == 0:
                print('')
                print('Goodbye!')
                return

        elif (choice == 2) and encode_or_decode(selected_file) == "encode":
            encode(selected_file, words, codes, abstr_freqs,
                         total_freqs, abstracts, encodedAbstracts)
            write_encoded_files(selected_file, words, codes, abstr_freqs,
                         total_freqs, abstracts, encodedAbstracts)
            print("")
            print_encode_result(selected_file, words, codes,
                             total_freqs, abstracts)
            print()
            print('Thank you, come again.')
            return
        elif (choice == 2) and encode_or_decode(selected_file) == "decode" :
            decode(selected_file , words, codes, abstr_freqs ,
                        total_freqs , abstracts , encodedAbstracts)
            write_decoded_files(selected_file , words, codes, abstr_freqs ,
                      total_freqs , abstracts , encodedAbstracts)
            print()
            print('Thank you, come again.')
            return

        elif choice == 0:
            print('')
            print('Goodbye!')
            return

def store_word(word, abstract_nr , abstr_freqs , total_freqs , codes, words): #t

    if not word in codes:
        wordCount = len(words) + 1
        words[wordCount] = word
        codes[word] = wordCount

    myWord = word.lower()

    if not myWord in total_freqs:
        total_freqs[myWord] = 1
    else:
        total_freqs[myWord] += 1

    if not abstract_nr in abstr_freqs:
        abstr_freqs[abstract_nr] = {}

    if not myWord in abstr_freqs[abstract_nr]:
        abstr_freqs[abstract_nr][myWord] = 1
    else:
        abstr_freqs[abstract_nr][myWord] += 1


def decode(file_name , words, codes,  abstr_freqs , total_freqs ,
        abstracts , encodedAbstracts):
    print("to do")

def encode(file_name , words, codes,  abstr_freqs , total_freqs ,
        abstracts , encodedAbstracts):
    word = ''
    abstractCount =  0
    #This is will check if a character is a \ or a number, which it will have
    file = open(file_name)
    for abstract in file.readlines():
        coded=''
        abstracts[abstractCount] = abstract;
        for char in abstract:
            if char.isalpha():
                word += char
                continue
            if char.isnumeric():
                coded += '\\' + char
                continue
            if char == '\\':
                coded += '\\' + char
                continue
            if word == '':
                coded += char
                continue

            store_word(word, abstractCount, abstr_freqs, total_freqs,
                       codes, words)
            coded += str(codes[word]) + char
            word =  ''

        if word != '':
            store_word(word, abstractCount, abstr_freqs, total_freqs,
                       codes, words)
            coded += str(codes[word])

        encodedAbstracts[abstractCount] = coded
        coded=''
        abstractCount += 1


    for index in words:
# Put all cap words without lowercase in codes with code 0
        word = words[index]
        low_word = word.lower()
        if low_word not in codes:
            codes[low_word] = 0


main('')
```