Lab 1 - Building dictionary with Selma Lagerlöf novels

Laboration 1 in EDAN20 @ LTH - http://cs.lth.se/edan20/coursework/assignment-1/ (http://cs.lth.se/edan20/coursework/assignment-1/)

Authour: Jonatan Kronander - elt15jkr@student.lu.se

The objectives of this assignment are to:

-Write a program that collects all the words from a set of documents

-Build an index from the words

-Know what indexing is

-Represent a document using the Tf.Idf value

-Write a short report of 1 to 2 pages on the assignment

-Read a short text on an industrial system

Indexing one file

- The index file will contain all the unique words in the document, where each word is associated with the list of its positions in the document.
- You will represent this index as a dictionary where the keys will be the words and the values, the lists of positions

- As words, you will consider all the strings of letters that you will set in lower case. You will not index the rest (i.e. numbers or symbols).

This is done by first using function txtClean.

In [5]:
```
import regex as re

def txtClean(text):
    """
    Replace capital characters to small characters

    Input txt file
    Output txt file
    """
    # Remove new lines
    text = re.sub("\n", " ", text)

    # Replace [A-Ö] with [a-ö]
    def toLowercase(matchobj):
        return matchobj.group(1).lower()

    text = re.sub(r'([A-Z])', toLowercase, text)

    # Remove multiple spaces
    text = re.sub(' +', ' ', text)

    return text
```

In [6]:
```
text = open("Selma/bannlyst.txt").read()
txt = txtClean(text)
```

- To extract the words, you will use Unicode regular expressions. Do not use \w+, for instance, but the Unicode equivalent. The word positions will correspond to the number of characters from the beginning of the file. (The word offset from the beginning)
- You will use finditer() to find the positions of the words. This will return you match objects, where you will get the matches and the positions with the group() and start() methods.

In [7]:
```
def string2dict(text,originaltext):
    """
    Creates a dict with (word:list[index apperences]) from input string

    Input string, string
    Output dict
    """
    stringList = re.findall(r"[a-zåäö]+",text) #This finds all words from a txt file. r"[a-zåäö]+ equal to r"\w+"

    stringDict = dict.fromkeys(stringList) #Creates dict (and remove doublicates)

    for word in stringDict:
        wordIndices = []
        pattern = r"\b"+word+ r"\b" #Only look at word

        for m in re.finditer(pattern, originaltext): #Iterate thorugh every word
            wordIndices.append(m.start(0))

        stringDict.update({word:wordIndices})
    return stringDict
```

In [36]:
```
txtDict = string2dict(txt,text)
```

In [37]:
```
len(txtDict)
```

Out[37]:
```
7950
```

Test with bannlyst text

The word gjord occurs three times in the text at positions 8551, 183692, and 220875, uppklarnande, once at position 8567, and stjärnor, once at position 8590.

In [38]:
```
txtDict['gjord']
```

Out[38]:
```
[8551, 183692, 220875]
```

In [39]:
```
txtDict['uppklarnande']
```

Out[39]:
```
[8567]
```

In [40]:
```
txtDict['stjärnor']
```

Out[40]:
```
[8590]
```

Pickle

- You will use the pickle package to write your dictionary in an file, see https://wiki.python.org/moin/UsingPickle (https://wiki.python.org/moin/UsingPickle).

In [8]:

```
import pickle

#with open('BannlystTxtDict.pickle', 'wb') as handle:
#    pickle.dump(txtDict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Open pickle

In [9]:

```
with open('BannlystTxtDict.pickle', 'rb') as handle:
    BannlystTxtDict = pickle.load(handle)
```

Test of pickle

In [12]:

```
BannlystTxtDict == txtDict
```

Out[12]:

```
True
```

Reading the content of a folder

Write a function that reads all the files in a folder with a specific suffix (txt). You will need the Python os package, see https://docs.python.org/3/library/os.html (https://docs.python.org/3/library/os.html). You will return the file names in a list.

Use function:

In [10]:

```
import os

def get_files(fileDir, suffix):
    """
    Returns all the files in a folder ending with suffix
    :param filedir:
    :param suffix:
    :return: the list of file names
    """
    files = []
    for file in os.listdir(fileDir):
        if file.endswith(suffix):
            files.append(file)
    return files
```

In [11]:

```
files = get_files("selma", ".txt")
```

In [12]:

```
files
```

Out[12]:

```
['troll.txt',
 'kejsaren.txt',
 'marbacka.txt',
 'herrgard.txt',
 'nils.txt',
 'osynliga.txt',
 'jerusalem.txt',
 'bannlyst.txt',
 'gosta.txt']
```

Creating a master index

Complete your program with the creation of master index, where you will associate each word of the corpus with the files, where it occur and its positions. (a posting list)

In [13]:

```
def toLowercase(matchobj):
    return matchobj.group(1).lower()

def addAll(fileDir,files):
    """
    This function takes way to long. Do not iterate word in dict but build dict directly.

    Reads all files in list and matches to txt files
    :param dir:
    :param files:
    :return dict:
    """
    allDict = {}

    for file in files:
        text = open(fileDir+"/"+file).read()
        txt = txtClean(text)
        stringList = re.findall(r"[a-zåäö]+",txt)
        allDict.update(dict.fromkeys(stringList)) # is this ok?!

    wordIndices = []
    i = 0

    for word in allDict: # Iterate through every word in master dict
        pattern = r"\b"+word+ r"\b" # Only look at word
        allDict[word] = {}

        for file in files: # Iterate through every text
            text = re.sub(r'([A-Z])', toLowercase, open(fileDir+"/"+file).read()) # Open text and lowercase all

            for m in re.finditer(pattern, text): # Iterate through every word in file text
                wordIndices.append(m.start(0))

            allDict[word][file] = list(wordIndices)
            wordIndices.clear()

        print(i/len(allDict)) # print finish procent
        i = i+1

    return allDict
```

In [ ]:

```
masterDict = addAll('selma',files)
```

In [220]:

```
#with open('masterDict1.pickle', 'wb') as handle:
#    pickle.dump(masterDict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [14]:

```
with open('masterDict1.pickle', 'rb') as handle:
    masterDict = pickle.load(handle)
```

Test of master dict. Below is an except of the master index with the words samlar and ände:

'samlar': {'nils.txt': [53499, 120336], 'gosta.txt': [317119, 414300, 543686],'osynliga.txt': [410995, 871322]},

In [15]:

```
masterDict["samlar"]
```

Out[15]:

```
{'troll.txt': [],
 'kejsaren.txt': [],
 'marbacka.txt': [],
 'herrgard.txt': [],
 'nils.txt': [53499, 120336],
 'osynliga.txt': [410995, 871322],
 'jerusalem.txt': [],
 'bannlyst.txt': [],
 'gosta.txt': [317119, 414300, 543686]}
```

'ände':{'nils.txt': [3991],'kejsaren.txt': [51100],'marbacka.txt': [374231],'troll.txt': [39726],'osynliga.txt': [742747]},

In [16]:

```
masterDict["ände"]
```

Out[16]:

```
{'troll.txt': [39726],
 'kejsaren.txt': [51100],
 'marbacka.txt': [374231],
 'herrgard.txt': [],
 'nils.txt': [3991],
 'osynliga.txt': [],
 'jerusalem.txt': [],
 'bannlyst.txt': [],
 'gosta.txt': []}
```

Representing Documents with tf-idf

Once you have created the index, you will represent each document in your corpus as a word vector. You will define the value of a word in a document with the tf-idf metric. Tf will be the relative frequency of the term in the document and idf, the logarithm base 10 of the inverse document frequency.

In [17]:

```
import math

def tiIdf(masterDict):
    """
    This function takes way to long. Do not iterate word in dict but build dict directly.

    Creates a ft-idf dict from all files.
    https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/
    :Param dict:
    :return dict:
    """
    tfIdfDict = masterDict.copy()
    j = 0

    lenText = {}
    for file in masterDict['nils']: # Read total nbr of words in each text
        text = open('selma'+"/"+file).read()
        txt = txtClean(text)
        stringList = re.findall(r"[a-zåäö]+",text)
        lenText[file] = len(stringList) # nbr of words in textfile

    for word in masterDict:
        #idf will be the logarithm base 10 of the inverse document frequency.
        nbrKeys = len(masterDict[word].keys())
        dictValues = masterDict[word].values()
        lenDictValues = len(dictValues)

        i = 0
        for fileList in dictValues: # Count nbr of empty list. (There is probably a better way to do this)
            if not fileList:
                i = i + 1

        df = (lenDictValues-i)
        if df == 0: #?! Why is df somethimes = 0. Indicates that there are problem in function addAll()
            idf = 0
        else:
            idf = math.log10(nbrKeys/df)

        for file in masterDict[word]:
            # Tf will be the relative frequency of the term in the document
            lenWordVec = len(masterDict[word][file]) # nbr of occurencies of word

            tf =  lenWordVec / lenText[file]
            tfIdfDict[word][file] = tf*idf

    return tfIdfDict
```

In [ ]:

```
tfIdfDict = tiIdf(masterDict)
```

In [ ]:

```
#with open('tfIdfDict.pickle', 'wb') as handle:
#    pickle.dump(tfIdfDict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [18]:

```
with open('tfIdfDict.pickle', 'rb') as handle:
    tfIdfDict = pickle.load(handle)
```

Test of tf idf:

känna :: bannlyst.txt 0.0, gosta.txt 0.0, herrgard.txt 0.0, jerusalem.txt 0.0, nils.txt 0.0

In [19]:

```
tfIdfDict['känna']
```

Out[19]:

```
{'troll.txt': 0.0,
 'kejsaren.txt': 0.0,
 'marbacka.txt': 0.0,
 'herrgard.txt': 0.0,
 'nils.txt': 0.0,
 'osynliga.txt': 0.0,
 'jerusalem.txt': 0.0,
 'bannlyst.txt': 0.0,
 'gosta.txt': 0.0}
```

gås :: bannlyst.txt 0.0, gosta.txt 0.0, herrgard.txt 0.0, jerusalem.txt 0.0, nils.txt 0.00010123719421964931

In [20]:

```
tfIdfDict['gås']
```

Out[20]:

```
{'troll.txt': 0.0,
 'kejsaren.txt': 0.0,
 'marbacka.txt': 0.0,
 'herrgard.txt': 0.0,
 'nils.txt': 0.00010139137475638062,
 'osynliga.txt': 0.0,
 'jerusalem.txt': 0.0,
 'bannlyst.txt': 0.0,
 'gosta.txt': 0.0}
```

nils :: bannlyst.txt 0.0, gosta.txt 0.0, herrgard.txt 0.0 jerusalem.txt 4.778415355159037e-06, nils.txt 9.801209641132888e-05

In [21]:

```
tfIdfDict['nils']
```

Out[21]:

```
{'troll.txt': 3.6624988178210027e-06,
 'kejsaren.txt': 8.107749884176785e-06,
 'marbacka.txt': 7.5972829304119305e-06,
 'herrgard.txt': 0.0,
 'nils.txt': 9.81613652422891e-05,
 'osynliga.txt': 0.0,
 'jerusalem.txt': 4.784864200624293e-06,
 'bannlyst.txt': 0.0,
 'gosta.txt': 0.0}
```

et :: bannlyst.txt 6.2846093167673765e-06, gosta.txt 0.0, herrgard.txt 0.0, jerusalem.txt 0.0, nils.txt 0.0

In [22]:

```
tfIdfDict['et']
```

Out[22]:

```
{'troll.txt': 0.0,
 'kejsaren.txt': 6.061569061072415e-05,
 'marbacka.txt': 1.4199826035912037e-05,
 'herrgard.txt': 0.0,
 'nils.txt': 0.0,
 'osynliga.txt': 0.0,
 'jerusalem.txt': 0.0,
 'bannlyst.txt': 6.2943926164517935e-06,
 'gosta.txt': 0.0}
```

Comparing Documents

Using the cosine similarity, compare all the pairs of documents with their tfidf representation and present your results in a matrix. You will include this matrix in your report.

Give the name of the two novels that are the most similar.

There are the document representations in term of words. Rows: documents, Col: words.

In [23]:

```
import numpy as np

docMatrix = np.zeros((9,len(tfIdfDict.keys())))
wordList = tfIdfDict.keys()
fileList = tfIdfDict['nils']

for i, word in enumerate(wordList):
    for j, file in enumerate(fileList):
        docMatrix[j,i] = tfIdfDict[word][file]
```

In [25]:

```
import numpy as np; import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

df = pd.DataFrame(docMatrix)

simularityMatrix = cosine_similarity(df)

print()

index = simularityMatrix[:,:].flatten().argsort()[-10:][::-1][9] # This is the

print("Position: (", index//9, ",", index, ") has maximum val")

print("That correspond to text:", list(fileList.keys())[index//9], "and", list(fileList.keys())[index], "Hence they are most similar with cosine value:", simularityMatrix[0,1
])
```

```
Position: ( 0 , 1 ) has maximum val
That correspond to text: troll.txt and kejsaren.txt Hence they are most similar with cosine value: 0.089223020097
```

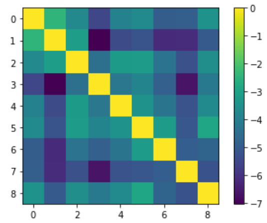
Plot matrix

In [31]:

```
import matplotlib.pyplot as plt

plt.imshow(np.log(simularityMatrix))
plt.colorbar()
plt.show()

for i,file in enumerate(list(fileList.keys())):
    print(i, "=", file)
```



```
0 = troll.txt
1 = kejsaren.txt
2 = marbacka.txt
3 = herrgard.txt
4 = nils.txt
5 = osynliga.txt
6 = jerusalem.txt
7 = bannlyst.txt
8 = gosta.txt
```

```
In [26]: for i,iFile in enumerate(list(fileList.keys())):
             for j, jFile in enumerate(list(fileList.keys())):
                 print(iFile, jFile, simularityMatrix[i,j])
```

```
troll.txt troll.txt 1.0
troll.txt kejsaren.txt 0.089223020097
troll.txt marbacka.txt 0.0240253830604
troll.txt herrgard.txt 0.00397627174328
troll.txt nils.txt 0.0197380431181
troll.txt osynliga.txt 0.0253165892183
troll.txt jerusalem.txt 0.00709510856874
troll.txt bannlyst.txt 0.00738801070435
troll.txt gosta.txt 0.0321187612557
kejsaren.txt troll.txt 0.089223020097
kejsaren.txt kejsaren.txt 1.0
kejsaren.txt marbacka.txt 0.0462649676222
kejsaren.txt herrgard.txt 0.000893781944723
kejsaren.txt nils.txt 0.00454629096198
kejsaren.txt osynliga.txt 0.00554194108819
kejsaren.txt jerusalem.txt 0.00207632604704
kejsaren.txt bannlyst.txt 0.00218541325775
kejsaren.txt gosta.txt 0.00658179194216
marbacka.txt troll.txt 0.0240253830604
marbacka.txt kejsaren.txt 0.0462649676222
marbacka.txt marbacka.txt 1.0
marbacka.txt herrgard.txt 0.0117883250111
marbacka.txt nils.txt 0.0409181304811
marbacka.txt osynliga.txt 0.0422248576751
marbacka.txt jerusalem.txt 0.0129240862132
marbacka.txt bannlyst.txt 0.00513431573213
marbacka.txt gosta.txt 0.0287715919395
herrgard.txt troll.txt 0.00397627174328
herrgard.txt kejsaren.txt 0.000893781944723
herrgard.txt marbacka.txt 0.0117883250111
herrgard.txt herrgard.txt 1.0
herrgard.txt nils.txt 0.0143551403561
herrgard.txt osynliga.txt 0.0212914838078
herrgard.txt jerusalem.txt 0.00764883514131
herrgard.txt bannlyst.txt 0.00129647328026
herrgard.txt gosta.txt 0.0151995600643
nils.txt troll.txt 0.0197380431181
nils.txt kejsaren.txt 0.00454629096198
nils.txt marbacka.txt 0.0409181304811
nils.txt herrgard.txt 0.0143551403561
nils.txt nils.txt 1.0
nils.txt osynliga.txt 0.0325868323216
nils.txt jerusalem.txt 0.013274155033
nils.txt bannlyst.txt 0.00543304080916
nils.txt gosta.txt 0.0254165000011
osynliga.txt troll.txt 0.0253165892183
osynliga.txt kejsaren.txt 0.00554194108819
osynliga.txt marbacka.txt 0.0422248576751
osynliga.txt herrgard.txt 0.0212914838078
osynliga.txt nils.txt 0.0325868323216
osynliga.txt osynliga.txt 1.0
osynliga.txt jerusalem.txt 0.041886005237
osynliga.txt bannlyst.txt 0.00598408045004
osynliga.txt gosta.txt 0.055504303364
jerusalem.txt troll.txt 0.00709510856874
jerusalem.txt kejsaren.txt 0.00207632604704
jerusalem.txt marbacka.txt 0.0129240862132
jerusalem.txt herrgard.txt 0.00764883514131
jerusalem.txt nils.txt 0.013274155033
jerusalem.txt osynliga.txt 0.041886005237
jerusalem.txt jerusalem.txt 1.0
jerusalem.txt bannlyst.txt 0.00731671952295
jerusalem.txt gosta.txt 0.00855904659069
bannlyst.txt troll.txt 0.00738801070435
bannlyst.txt kejsaren.txt 0.00218541325775
bannlyst.txt marbacka.txt 0.00513431573213
bannlyst.txt herrgard.txt 0.00129647328026
bannlyst.txt nils.txt 0.00543304080916
bannlyst.txt osynliga.txt 0.00598408045004
bannlyst.txt jerusalem.txt 0.00731671952295
bannlyst.txt bannlyst.txt 1.0
bannlyst.txt gosta.txt 0.00524144862443
gosta.txt troll.txt 0.0321187612557
gosta.txt kejsaren.txt 0.00658179194216
gosta.txt marbacka.txt 0.0287715919395
gosta.txt herrgard.txt 0.0151995600643
gosta.txt nils.txt 0.0254165000011
gosta.txt osynliga.txt 0.055504303364
gosta.txt jerusalem.txt 0.00855904659069
gosta.txt bannlyst.txt 0.00524144862443
gosta.txt gosta.txt 1.0
```

Reading

Read the text: Challenges in Building Large-Scale Information Retrieval Systems about the history of Google indexing by Jeff Dean. In your report, tell how your index encoding is related to what Google did. You must identify the slide where you have the most similar indexing technique and write the slide number in your report. https://static.googleusercontent.com/media/research.google.com/en//people/jeff/WSDM09-keynote.pdf (https://static.googleusercontent.com/media/research.google.com/en//people/jeff/WSDM09-keynote.pdf)

Answer:

On slide 14-17 they talk about ways of index partitioning and compare by doing so with doc or by word. Google and we have similar problem where we want to index shards partitioning from a set of documents. Google uses docs as index while we used word, where we had shard subset of words for all docs.